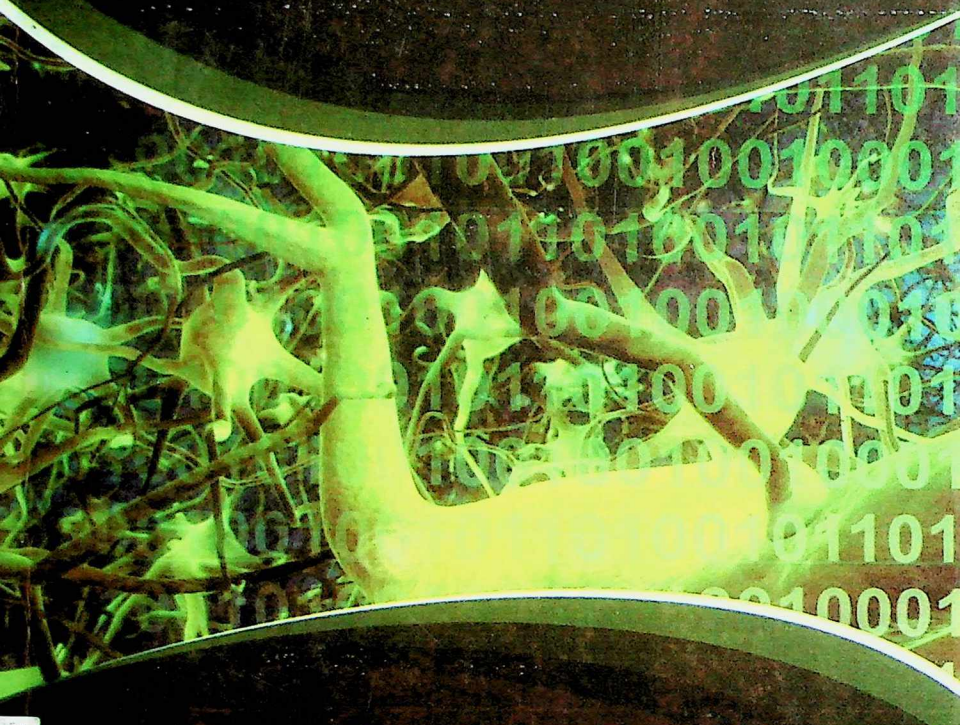
...thông tin - Thư viện
1162850

LÊ HOÀI BẮC
TÔ HOÀI VIỆT

CƠ SỞ TRÍ TUỆ NHÂN TẠO



NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

LÊ HOÀI BẮC - TÔ HOÀI VIỆT

Cơ sở
Trí tuệ nhân tạo

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
2014

Lời nói đầu

Với sự phát triển vượt bậc của công nghệ, máy tính cùng các thiết bị điện tử ngày càng đóng vai trò quan trọng trong mọi hoạt động của con người. Không chỉ hỗ trợ trong công việc, các máy tính ngày nay còn là một phương tiện hữu ích phục vụ cho các sinh hoạt hàng ngày của con người như hỗ trợ trong đi lại, cập nhật tin tức, giải trí... Tuy nhiên, để máy tính thật sự trở thành một người bạn thân thiết của con người, một số trở ngại cơ bản cần phải được vượt qua. Các máy tính ngày nay có sức mạnh vượt trội so với trước, khả năng xử lý và lưu trữ khổng lồ, nhưng nó vẫn chưa giải quyết được nhiều bài toán trên thực tế như: làm sao để có thể tự suy nghĩ và hành động trong những môi trường thế giới thực phức tạp, làm sao để học và rút trích các thông tin, tri thức từ môi trường xung quanh phục vụ cho hoạt động của mình... Giải quyết các vấn đề trên không chỉ đòi hỏi sức mạnh công nghệ mà còn cần đến những hướng tiếp cận, những thuật toán thông minh hơn.

Giáo trình này cung cấp các ý tưởng, kiến thức cơ bản về hướng nghiên cứu trí tuệ nhân tạo, ngành học nhằm mục tiêu thiết kế ra các phương pháp tính toán thông minh. Nội dung của giáo trình sẽ tập trung giải quyết những vấn đề cơ bản của của một hệ thống thông minh bao gồm: *các phương pháp tìm kiếm hiệu quả, biểu diễn tri thức và suy diễn, học máy*. Các nội dung trình bày trong giáo trình được tham khảo từ những giáo trình, tài liệu về trí tuệ nhân tạo phổ biến trên thế giới. Do đó, các kiến thức trong giáo trình phần nào cập nhật với những bước phát triển gần đây trong lĩnh vực nghiên cứu.

Với giáo trình này, chúng tôi hy vọng cung cấp cho người bạn một cái nhìn toàn cảnh và cơ bản về những nghiên cứu liên quan đến trí tuệ nhân tạo. Qua đó, có thể động viên, khuyến khích những ai có niềm đam mê nghiên cứu, sáng tạo tiếp tục tìm hiểu và phát triển các hướng chuyên sâu của ngành học. Nhóm biên soạn rất mong muốn bạn đọc tìm thấy những điều thú vị trong giáo trình này.

Chúng tôi xin trân trọng gửi lời cảm ơn đến Ban giám hiệu *Trường Đại học Khoa học Tự nhiên (Đại học Quốc gia Tp.HCM)*, các đồng nghiệp đang công tác tại *Khoa Công nghệ thông tin* và các trường bạn, *Nhà xuất*

bản Khoa học và Kỹ thuật đã đóng góp ý kiến và giúp đỡ chúng tôi trong quá trình biên soạn.

Do thời gian biên soạn có hạn, giáo trình không tránh khỏi những sai sót nhất định. Vì thế chúng tôi mong nhận được những ý kiến đóng góp của bạn đọc, nhằm giúp hoàn thiện giáo trình trong những lần tái bản tới. Chúng tôi rất mong nhận được những góp ý xây dựng từ quý độc giả để nâng cao chất lượng giáo trình trong những lần tái bản sau. Mọi góp ý xin gửi bằng thư hay *email* đến địa chỉ:

Khoa Công Nghệ Thông Tin,
Trường Đại học Khoa học tự nhiên, ĐHQG-TPHCM,
227 Nguyễn Văn Cừ, Quận 5, Thành phố Hồ Chí Minh,
Điện thoại: (08) 38354266
Email: fitbooks@fit.hcmus.edu.vn

Xin vui lòng ghi chủ đề "*Góp ý cho Giáo trình Cơ sở Trí tuệ Nhân tạo*".

Tp. Hồ Chí Minh, ngày 01 tháng 12 năm 2013,

Thay mặt nhóm biên soạn,

PGS. TS. Lê Hoài Bắc.

Mục lục

Lời nói đầu 1

Mục lục 3

Phần 1 GIẢI QUYẾT BÀI TOÁN TRÊN MÁY TÍNH BẰNG PHƯƠNG PHÁP TÌM KIẾM 10

Chương 1 CÁC CHIẾN LƯỢC TÌM KIẾM KHÔNG CÓ THÔNG TIN 11

I. BIỂU DIỄN BÀI TOÁN TÌM KIẾM 11

I.1 Đồ thị 12

I.2 Trạng thái 14

I.3 Hàm trạng thái con 14

I.4 Chi phí đường đi 14

I.5 Những bài toán tìm kiếm 15

II. TÌM THEO CHIỀU RỘNG (Breadth-first search - BFS) 20

II.1 Ý tưởng: Tìm các đường đi trong một số bước nhất định. 20

II.2 Con trỏ quay lui 26

II.3 Thuật toán tìm kiếm theo chiều rộng (BFS) 26

III. TÌM KIẾM CHI PHÍ ĐỒNG NHẤT (Uniform-cost Search - UCS) 27

III.1 Hàng đợi ưu tiên 28

III.2 Hàm chi phí 29

III.3 Thuật toán và điều kiện dừng. 29

III.4 Đánh giá thuật toán tìm kiếm 30

IV. TÌM THEO CHIỀU SÂU (Depth-first search - DFS) 34

IV.1 Tìm kiếm theo chiều sâu – Một thuật toán thay thế. 34

IV.2 Tránh lặp vô hạn – Các chiến lược kiểm tra 36
V. TRÁNH TRÙNG LẶP TRẠNG THÁI 41
VI. KẾT CHƯƠNG 44
VII. THUẬT NGỮ TIẾNG ANH 45
Chương 2 CÁC CHIẾN LƯỢC TÌM KIẾM CÓ THÔNG TIN 48
I. TÌM KIẾM CÓ THÔNG TIN 48
I.1 Hàm *heuristic* – Tìm kiếm tối ưu kiểu tham lam (Greedy Best-First search). 49
I.2 Tìm kiếm A* 51
II. HÀM *HEURISTIC*. 53
II.1 Ảnh hưởng của hàm *heuristic* lên hiệu quả 54
II.2 Phát minh ra các hàm *heuristic* chấp nhận được 56
III. TÌM KIẾM CỤC BỘ VÀ BÀI TOÁN TỐI ƯU 58
III.1 Tìm kiếm leo đồi 60
III.2 Tìm kiếm nhóm cục bộ 64
III.3 Tìm kiếm mô phỏng luyện kim 64
III.4 Thuật giải di truyền 66
IV. KẾT CHƯƠNG 69
V. THUẬT NGỮ TIẾNG ANH 70
Chương 3 BÀI TOÁN THỎA MÃN RÀNG BUỘC 72
I. BÀI TOÁN THỎA MÃN RÀNG BUỘC (CSP) 72
II. TÌM KIẾM QUAY LUI CHO CSP 76
III. TÌM KIẾM HEURISTIC CHO CSP 78
IV. KẾT CHƯƠNG 79
V. THUẬT NGỮ TIẾNG ANH 80
Phần 2 BIỂU DIỄN TRI THỨC VÀ SUY DIỄN 82

Chương 4 LOGIC VÀ SUY DIỄN 83
I. LOGIC 83
II. LOGIC MỆNH ĐỀ: LOẠI LOGIC ĐƠN GIẢN. 84
II.1 Cú pháp 84
II.2 Ngữ nghĩa 85
II.3 Tính tương đương, hợp lệ và thoả mãn được. 87
II.4 Bài toán suy dẫn trong logic mệnh đề. 88
III. CÁC CÁCH SUY DIỄN TRONG LOGIC MỆNH ĐỀ 89
III.1 Hợp giải 89
III.1.1 Dạng hội chuẩn 90
III.1.2 Thuật toán hợp giải, ... 91
III.2 Suy diễn tiến 93
III.3 Suy diễn lùi 96
IV. KẾT CHƯƠNG 96
V. THUẬT NGỮ TIẾNG ANH 97
Chương 5 LOGIC BẬC NHẤT 100
I. CÚ PHÁP VÀ NGỮ NGHĨA CỦA LOGIC BẬC NHẤT 100
I.1 Mô hình của logic bậc nhất 100
I.2 Ký hiệu và biểu diễn 101
I.3 Biểu thức 103
I.4 Câu nguyên tố 103
I.5 Câu phức 103
I.6 Lượng từ 104
I.7 Phép bằng 110
II. SỬ DỤNG LOGIC BẬC NHẤT 110
II.1 Ví dụ miền quan hệ "họ hàng" 110

II.2 Số, tập hợp và danh sách 114
II.3 Một số ví dụ chuyển câu sang logic bậc nhất 118
III. KẾT CHƯƠNG 122
IV. THUẬT NGỮ TIẾNG ANH 122
Chương 6 SUY DIỄN VỚI LOGIC BẬC NHẤT 124
I. SUY DIỄN MỆNH ĐỀ SO VỚI BẬC NHẤT 124
I.1 Luật suy diễn với lượng từ 124
I.2 Rút gọn về suy diễn mệnh đề 126
II. PHÉP ĐỒNG NHẤT 127
II.1 Luật suy diễn bậc nhất 127
II.2 Đồng nhất 128
III. SUY DIỄN TIẾN 130
III.1 Các mệnh đề xác định bậc nhất 131
III.2 Thuật toán suy diễn tiến đơn giản 132
IV. SUY DIỄN LÙI 134
IV.1 Thuật toán suy diễn lùi 135
IV.2 Lập trình logic 137
IV.3 Suy diễn dư thừa và lặp vô tận 139
V. HỢP GIẢI 140
V.1 Dạng hội chuẩn đối với logic bậc nhất 140
V.2 Luật suy diễn hợp giải 142
V.3 Các chiến lược hợp giải 147
VI. KẾT CHƯƠNG 148
VII. THUẬT NGỮ TIẾNG ANH 149
Chương 7 SUY DIỄN KHÔNG CHẮC CHẮN 151
I. GIỚI THIỆU VỀ SỰ KHÔNG CHẮC CHẮN 151

I.1 Sự không chắc chắn là gì? ... 151
I.2 Các nguyên nhân gây nên sự không chắc chắn ... 152
I.3 Các mô hình biểu diễn không chắc chắn ... 153
II. LÝ THUYẾT XÁC SUẤT CỦA *BAYES* ... 154
II.1 Mệnh đề ... 154
II.2 Sự kiện nguyên tố ... 155
II.3 Xác suất tiên nghiệm ... 156
II.4 Xác suất hậu nghiệm ... 157
III. SUY DIỄN *BAYES* ... 158
III.1 Suy diễn với bảng phân phối kết hợp ... 159
III.2 Luật *Bayes* ... 160
III.3 Ví dụ đơn giản ... 161
III.4 Kết hợp các chứng cứ ... 162
IV. LÝ THUYẾT HỆ SỐ CHẮC CHẮN VÀ SUY DIỄN DỰA TRÊN CHỨNG CỨ ... 163
IV.1 Hệ số chắc chắn ... 163
IV.2 Các phép toán ... 165
IV.3 Suy diễn dựa trên chứng cứ ... 166
V. SO SÁNH SUY DIỄN *BAYES* VÀ HỆ SỐ CHẮC CHẮN ... 169
VI. KẾT CHƯƠNG ... 171
VII. THUẬT NGỮ TIẾNG ANH ... 172
Phần 3 HỌC MÁY ... 174
Chương 8 GIỚI THIỆU HỌC MÁY ... 175
I. HỌC MÁY LÀ GÌ? ... 175
I.1 Các động lực của học máy ... 175
I.2 Các loại học ... 175

II. NAÏVE *BAYES* – MỘT MÔ HÌNH HỌC ĐƠN GIẢN 177
II.1 Ví dụ học máy 177
II.2 Mô hình xác suất 178
II.3 Luật *Bayes* 180
II.4 Phép sửa Laplace 182
III. HỌC CÂY QUYẾT ĐỊNH 183
III.1 Cây quyết định 184
III.2 Tính biểu diễn của cây quyết định 185
III.3 Rút cây quyết định từ ví dụ 186
III.4 Chọn thuộc tính kiểm tra 190
III.5 Đánh giá hiệu quả của thuật toán học 193
III.6 Nhiễu và quá khớp 193
III.7 Mở rộng tính ứng dụng của cây quyết định 195
IV. KẾT CHƯƠNG 196
V. THUẬT NGỮ TIẾNG ANH 197
Chương 9 MẠNG NƠ-RON NHÂN TẠO 198
I. GIỚI THIỆU – MẠNG NƠ-RON LÀ GÌ? 198
II. MẠNG NƠ-RON NHÂN TẠO 200
II.1 Kiến trúc mạng nơ-ron nhân tạo 200
II.2 Nơ-ron như là một thành phần tính toán đơn giản 201
III. PERCEPTRON 204
III.1 Các perceptron biểu diễn điều gì? 204
III.2 Thuật toán huấn luyện perceptron 206
III.3 Phân tích 207
IV. MẠNG NƠ-RON TRUYỀN THẲNG ĐA LỚP 210
IV.1 Kiến trúc mạng 210

IV.2 Thuật toán học lan truyền ngược 211
V. ỨNG DỤNG CỦA MẠNG NƠ-RON 218
VI. KẾT CHƯƠNG 220
VII. THUẬT NGỮ TIẾNG ANH 222
TÀI LIỆU THAM KHẢO 224
CHỈ MỤC THUẬT NGỮ 225

Phần 1

GIẢI QUYẾT BÀI TOÁN TRÊN MÁY TÍNH BẰNG PHƯƠNG PHÁP TÌM KIẾM

Chương 1

CÁC CHIẾN LƯỢC TÌM KIẾM KHÔNG CÓ THÔNG TIN

Nhiều bài toán đã được đưa vào máy tính để giải quyết, với mong muốn tận dụng được sức mạnh của máy tính để tìm lời giải nhanh và chính xác hơn. Đến nay, có nhiều phương pháp để giải một bài toán trên máy tính. Chương này và các chương sau của phần 1 giáo trình này sẽ trình bày về vấn đề giải quyết bài toán bằng các phương pháp tìm kiếm khác nhau. Điều kiện tiên quyết để giải quyết một bài toán là phải biểu diễn được bài toán đó. Do đó, phần đầu của chương này sẽ định nghĩa đầy đủ các thành phần của một bài toán và một số ví dụ minh họa kèm theo. Sau đó các thuật toán tìm kiếm để giải quyết bài toán sẽ được trình bày và so sánh với nhau. Những thuật giải này thuộc nhóm phương pháp tìm kiếm mù (tức là không có thêm bất kì thông tin nào khác ngoài các điều kiện của bài toán). Chương 2 đề cập đến những giải thuật tìm kiếm có thông tin bổ sung giúp xác định phương hướng tìm kiếm tốt hơn.

I. BIỂU DIỄN BÀI TOÁN TÌM KIẾM

Khi giải quyết một bài toán bằng phương pháp tìm kiếm (gọi tắt là bài toán tìm kiếm) trước hết phải biểu diễn được bài toán đó. Một bài toán thường gồm có vấn đề (yêu cầu) và lời giải cho bài toán. Phương pháp tìm kiếm được sử dụng để tìm ra lời giải trong không gian tìm kiếm. Biểu diễn một bài toán tìm kiếm thường bao gồm năm thành phần:

- Không gian trạng thái: Cho biết tất cả các trạng thái có thể có của bài toán.

- Trạng thái ban đầu.
  Nơi bắt đầu việc tìm kiếm. Ví dụ: Du khách sẽ xuất phát từ thành phố A.

- Hàm trạng thái con.
  Phát sinh trạng thái kế tiếp từ một trạng thái cụ thể x. Hàm trạng thái con, cùng với trạng thái ban đầu, tạo nên không gian trạng thái của bài toán. Không gian này bao gồm các trạng thái mà ta sẽ xem xét đến trong quá trình tìm kiếm.

- Một tập các trạng thái đích (kết thúc).
  Các trạng thái đích này đều thuộc không gian trạng thái. Tùy vào bài toán mà có một hay nhiều trạng thái đích khác nhau. Trong bài toán đi du lịch thì trạng thái đích chính là thành phố cần đến của du khách. Tuy nhiên, trong những bài toán khác thì trạng thái đích chỉ được mô tả bởi một vài điều kiện nào đó chứ không phải là một tập các trạng thái xác định nữa. Như trong cờ vua, trạng thái đích là tình huống "chiếu bí" tức là tình huống đối phương không thể ngăn cản được việc quân vua của mình bị tấn công.

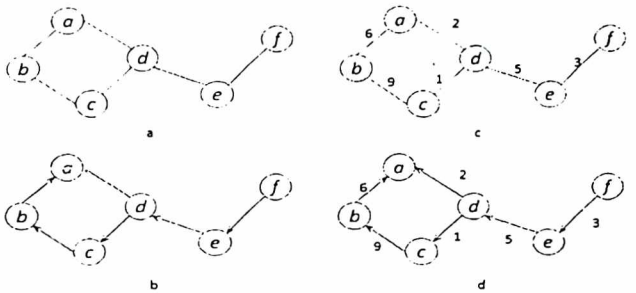
- Hàm chi phí đường đi.
  Dùng để tính chi phí cho lời giải của bài toán.

Với cách biểu diễn này ta có thể chuyển bài toán tìm lời giải về bài toán tìm đường đi từ trạng thái ban đầu đến trạng thái đích.

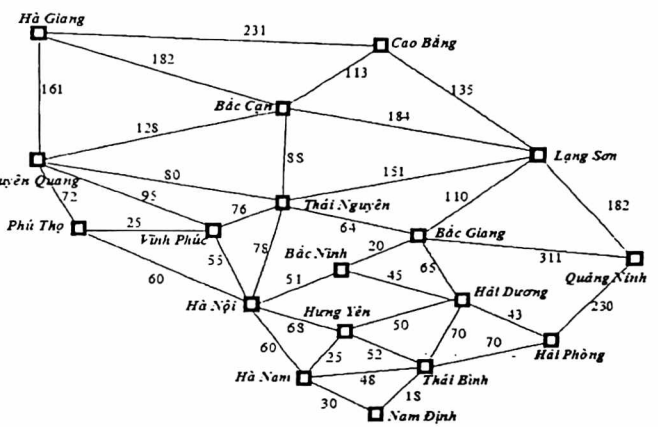
I.1 Đồ thị

Để giải quyết tốt một bài toán tìm kiếm cần phải xác định được không gian trạng thái phù hợp. Dựa vào trạng thái ban đầu và hàm trạng thái con chúng ta sẽ xác định được không gian trạng thái của bài toán tìm kiếm, trong đó không gian trạng thái được hiểu là tập các trạng thái có thể đến được từ trạng thái ban đầu. Không gian trạng thái này thường được biểu diễn dưới dạng một đồ thị (gọi là *đồ thị không gian trạng thái* hoặc *đồ thị trạng thái*). Đồ thị là một tập hợp các đỉnh được nối với nhau bởi các cung, các cung nối này có thể có hướng hoặc vô hướng, có trọng số hoặc không có trọng số (Hình 1.1). Mỗi đỉnh trong đồ thị sẽ tương ứng với một trạng thái và cung nối giữa các đỉnh thể

hiện có một hành động/phép đổi từ một trạng thái (đỉnh) này sang một trạng thái (đỉnh) khác. Giả sử, xét bài toán một người đi du lịch tìm một lộ trình giữa các tỉnh-thành phố ở vùng Đông Bắc Bộ. Hình 1.2 thể hiện bản đồ các thành phố và cũng được xem là một đồ thị trạng thái với các đỉnh là các thành phố và các cung chính là đường đi hai chiều từ thành phố này sang thành phố khác.



Hình 1.1 - *Ví dụ về đồ thị*: a. *Đồ thị vô hướng*. b. *Đồ thị có hướng*. c. *Đồ thị vô hướng và có trọng số*. d. *Đồ thị có hướng và có trọng số*.



Hình 1.2 - *Bản đồ vùng Đông Bắc Bộ.*

I.2 Trạng thái

Trạng thái trong bài toán tìm kiếm chính là thể hiện của một giai đoạn trong quá trình giải bài toán này. Một trạng thái bao gồm những thông tin về thế giới trong bài toán tìm kiếm.Ví dụ: Bài toán 8-puzzle (Hình 1.3) thế giới của bài toán chỉ giới hạn trong một bàn cờ kích thước 3×3 với 8 quân cờ và một trạng thái chỉnh là một cách sắp xếp 8 quân cờ trên bàn cờ 3×3. Đối với bài toán tìm đường, để đi từ thành phố này đến thành phố khác có thể phải đi qua nhiều thành phố trung gian, trạng thái của bài toán này cho biết đang ở thành phố nào.

I.3 Hàm trạng thái con

Hàm này thông thường được dùng để mô tả những biến đổi hay hành động có thể xảy ra cho một trạng thái nào đó. Cho trước trạng thái x cụ thể, hàm **Con**(x) sẽ trả về các cặp (**hành động**, **trạng thái con**), trong đó **hành động** là một trong những hành động hợp lệ trên trạng thái x và **trạng thái con** là trạng thái phát sinh được từ x bởi hành động tương ứng đó. Ví dụ: với trạng thái **Ở**(Hà Giang) hàm trạng thái con trong bài toán tìm đường đi trong các thành phố trong Hình 1.2 sẽ là:

Con(***Ở***(Hà Giang)) = {(***Đi***(*Tuyên Quang*),***Ở***(*Tuyên Quang*)),(***Đi***(*Bắc Cạn*),***Ở***(*Bắc Cạn*)),(***Đi***(*Cao Bằng*),***Ở***(*Cao Bằng*))}.

I.4 Chi phí đường đi

Một lời giải (*đường đi*) là một chuỗi các trạng thái liên kết với nhau qua các thao tác biến đổi. Mỗi lời giải cho bài toán đều có chi phí xác định. Chi phí ở đây dùng để đánh giá hiệu quả của giải thuật tìm kiếm cũng như là độ tốt của chính lời giải đó. Chẳng hạn với bài toán người người bán hàng (mô tả ở phần I.5 chương 1) thì chi phí được tính bằng chiều dài theo đơn vị ki-lô-mét. Một cách tổng quát thì chi phí cho một lời giải được tính bằng tổng chi phí của từng bước thực hiện của lời giải đó, với giả sử là chi phí của từng bước không âm. Chi phí cho bước biến đổi a chuyển từ trạng thái x sang trạng thái y được kí hiệu là: **Chi_phí_Bước**(a, x, y), có thể viết gọn thành: **Chi_phí_Bước**(x, y) nếu chỉ có duy nhất một thao tác biến đổi a.

I.5 Những bài toán tìm kiếm

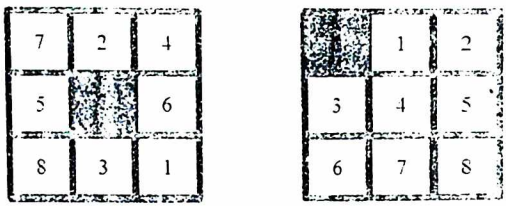
Giải quyết bài toán bằng phương pháp tìm kiếm được áp dụng trên nhiều lĩnh vực khác nhau. Ở đây, giáo trình liệt kê một số bài toán tìm kiếm thông dụng, phân thành hai nhóm: *bài toán trò chơi* và *bài toán thực tế*. Bài toán trò chơi minh họa cho những phương pháp giải quyết vấn đề khác nhau. Thông thường loại bài toán này luôn có mô tả chính xác, cụ thể. Có điểm chung thống nhất như vậy giúp dễ dàng so sánh hiệu quả của các thuật toán khác nhau. Bài toán thực tế lại được người ta quan tâm nhiều hơn. Tuy nhiên, những bài toán loại này thường không có một mô tả thống nhất nào, mà chỉ có dạng phát biểu tổng quát ban đầu.

Bài toán trò chơi

8-puzzle

Bài toán 8-puzzle (còn gọi là Taci) thường được sử dụng để thử nghiệm các thuật toán tìm kiếm. Trong bài toán này, chúng ta có một bàn cờ kích thước 3×3 trong đó 8 quân cờ được đánh số từ 1 đến 8 và đặt ở 8 ô vuông, ô còn lại để trống. Các quân cờ có thể được đẩy vào ô trống kế cận nó. Mục tiêu của trò chơi này là đẩy các quân cờ sao cho từ cách sắp xếp ban đầu (trạng thái đầu) đưa về được cách sắp xếp mong muốn (trạng thái đích) như trong Hình 1.3. Bài toán được phát biểu/biểu diễn lại đầy đủ như sau:

- **Trạng thái**: Một cách đặt 8 quân cờ vào 9 ô vuông.
- **Trạng thái ban đầu**: Tất cả trạng thái đều có thể là trạng thái đầu, tuy nhiên không phải trạng thái nào cũng tới được trạng thái đích xác định trước.
- **Hàm trạng thái con**: Phát sinh một trạng thái mới bằng cách di chuyển các quân (hoặc ô trống). Có bốn thao tác di chuyển: trái, phải, lên và xuống.
- **Chi phí đường đi**: Mỗi bước đều có chi phí bằng 1 nên chi phí sẽ bằng số bước thực hiện.

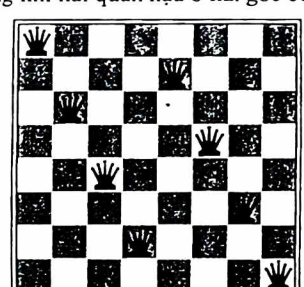


Hình 1.3 - *Một dạng của bài toán 8-puzzle.*

Bài toán 8-puzzle thuộc nhóm trò chơi khối trượt. Bài 8-puzzle có: 9!/2 = 181.440 trạng thái có thể và việc giải bài này là khá dễ. Với 15-puzzle (trên bàn cờ 4×4) có đến 1.3 nghìn tỉ trạng thái và thuật toán tìm kiếm tốt nhất giải ra chỉ trong vòng vài mili giây. Nhưng 24-puzzle (bàn cờ 5×5) với khoảng 10^{25} trạng thái thì khó có thể giải quyết một cách hiệu quả với các thuật toán hiện nay.

Bài toán 8 quân hậu

Mục đích của bài toán 8 quân hậu là đặt 8 quân hậu trên bàn cờ vua, sao cho không có con hậu nào tấn công được nhau. (Một quân hậu có thể tấn công theo: dòng, cột và đường chéo). Hình 1.4 cho thấy một cách đặt không đúng khi hai quân hậu ở hai góc có thể ăn được nhau.



Hình 1.4 - *Một lời giải gần đúng của bài toán 8 hậu.*

Mặc dù có một cách giải đặc biệt cho riêng bài toán này cũng như cho lớp bài toán *n* quân hậu, giải quyết bài toán bằng phương pháp tìm kiếm cũng có nhiều điều thú vị. Có hai cách biểu diễn bài toán này, *một là* theo dạng ta đặt từng quân hậu lên bàn cờ và khởi đầu bằng một bàn cờ trống, *hai là* bắt đầu với cả 8 quân hậu trên bàn cờ và di chuyển những quân hậu này. Trong cả hai cách đã nêu thì chi phí không được quan tâm, mà chỉ có trạng thái đích được quan tâm. Cách biểu diễn đầu tiên được biểu diễn như sau:

- **Trạng thái:** Một cách sắp xếp từ 0 đến 8 hậu trên bàn cờ.
- **Trạng thái ban đầu:** Không có quân hậu nào trên bàn cờ
- **Hàm trạng thái con:** Phát sinh một trạng thái mới bằng cách thêm một quân hậu vào bàn cờ.
- **Trạng thái đích:** Tất cả 8 quân hậu trên bàn cờ đều không ăn được nhau.
- **Chi phí đường đi:** Bằng 0 cho mỗi bước thực hiện.

Số trường hợp phải xét sẽ rất lớn ($64.64...57 \approx 1.8 \times 10^{14}$) nếu không loại bỏ việc đặt vào những vị trí quân hậu đó có thể tấn công. Với cách này ta có thể biểu diễn lại như sau:

- **Trạng thái:** Một cách sắp xếp n quân hậu (với $0 \leq n \leq 8$) mỗi quân ở một cột tính từ bên trái qua, sao cho không có quân nào tấn công được nhau.
- **Hàm trạng thái con:** Phát sinh một trạng thái mới bằng cách thêm một quân hậu vào cột trống đầu tiên tính từ bên trái, sao cho không có quân hậu nào tấn công được nhau.

Cách biểu diễn mới này đã làm giảm không gian trạng thái từ: 3×10^{14} xuống còn 2057 trạng thái, như vậy sẽ rất dễ cho việc tìm lời giải. Tuy nhiên, với 100 quân hậu thì mặc dù đã giảm số lượng trạng thái xuống khá nhiều, nhưng những thuật toán trong phạm vi chương này vẫn chưa thể giải quyết được.

Bài toán thực tế

Bài toán tìm đường đi

Bài toán tìm đường đi có thể sử dụng đồ thị để định nghĩa. Bài toán này thường được ứng dụng trong nhiều lĩnh vực khác như: định tuyến trong mạng máy tính, hệ thống lập kế hoạch các chuyến bay. Những vấn đề này thường khó mô tả. Một ví dụ đơn giản bài toán du lịch hàng không được mô tả như sau:

- **Trạng thái:** Thể hiện bởi địa điểm (sân bay) và thời điểm (giờ) hiện tại.
- **Trạng thái ban đầu:** Được mô tả bởi bài toán.
- **Hàm trạng thái con:** Trả về trạng thái kết quả của việc đi lên một chuyến bay theo lịch (gồm cả loại ghế và vị trí) bay từ sân bay này đến sân bay khác, thời điểm đến sẽ tính thêm khoảng thời gian cất cánh trễ (nếu có) và khoảng thời gian di chuyển lên máy bay.
- **Trạng thái đích:** Điểm đến tại một thời điểm xác định.
- **Chi phí đường đi:** Phụ thuộc vào tiền vé, thời gian chờ, thời gian bay, thủ tục nhập cảnh, loại ghế, loại máy bay v.v...

Hệ thống tư vấn du lịch sử dụng phát biểu bài toán dạng như vậy, cùng với những điều kiện phức tạp khác. Những người thường xuyên đi máy bay đều biết không phải chuyến bay nào cũng đúng lịch trình đã định. Một hệ thống là thực sự tốt khi có thêm những kế hoạch đối phó với những sự cố như: đặt vé dự phòng trên những chuyến bay khác dựa trên chi phí và khả năng thất bại của kế hoạch ban đầu.

Bài toán người đi du lịch

Cũng thuộc loại bài toán tìm đường đi nhưng lại khó hơn. Chẳng hạn, với yêu cầu là du khách sẽ xuất phát từ một thành phố đi thăm hết các thành phố khác ít nhất một lần, rồi trở về nơi xuất phát. Cũng giống như bài toán tìm đường đi, hành động phát sinh trạng thái là di chuyển giữa hai thành phố liền kề nhau. Tuy nhiên, không gian trạng thái lại khác. Mỗi trạng thái bây giờ có thêm tập các thành phố đã đi qua.

Trạng thái đầu sẽ có dạng "Ở A; đi qua{A}", một trạng thái trung gian có dạng là "Ở E; đi qua{A, B, C, D}". Trạng thái đích sẽ là đang ở tại thành phố ban đầu và tất cả thành phố đều đã được đi qua.

Bài toán người bán hàng

Một trường hợp đặc biệt của bài toán người đi du lịch với yêu cầu cao hơn là mỗi thành phố chỉ được đi qua một lần. Mục đích là tìm đường đi ngắn nhất thỏa mãn điều kiện này. Bên cạnh ứng dụng lập kế hoạch cho người bán hàng bài toán này còn được ứng dụng cho các công việc như: lập kế hoạch cho mũi hàn mạch điện tử tự động và máy xếp hàng trong siêu thị.

Bài toán bố trí mạch điện tử

Bài toán này yêu cầu bố trí hàng triệu thành phần và mối nối trên một con chip sao cho tối thiểu hóa diện tích, độ trễ, mất mát và tối đa hóa lợi nhuận cho nhà sản xuất. Vấn đề bố trí này đi sau công việc thiết kế, thông thường gồm hai phần: bố trí các ô và định tuyến kênh kết nối. Trong bố trí ô, mỗi thành phần chính của mạch được gom lại thành các ô, mỗi ô thực hiện một chức năng khác nhau. Mỗi ô có một dạng nhất định và cần một số lượng đường kết nối nhất định. Mục đích ở phần này là đặt các ô trên con vi mạch sao cho nó không chồng lên nhau cũng như có chỗ trống cho đường nối giữa các ô. Phần thứ hai sẽ tìm đường đi cho các mạch nối qua khoảng trống giữa các ô. Dạng bài toán tìm kiếm này cực kì khó nhưng lại rất cần trong thực tế.

Bài toán robot tìm đường

Bài toán tổng quát của bài toán tìm đường đi. Trong đó, thay vì tập rời rạc các đường đi thì robot có thể di chuyển trong một không gian liên tục (trên lí thuyết) với một tập các hành động và trạng thái không giới hạn. Đối với robot có bánh di chuyển trên mặt phẳng thì không gian là hai chiều. Khi robot có thêm tay chân thì không gian sẽ trở thành nhiều chiều hơn. Bài toán này không chỉ đơn thuần là tìm kiếm, trên thực tế còn có những xử lý khác như: xử lý lỗi tín hiệu cảm biến.

Bài toán dây chuyền lắp ráp tự động

Những thiết bị phức tạp được lắp ráp bởi robot được trình diễn đầu tiên bởi Freddy năm 1972. Từ đó quá trình phát triển tuy chậm mà

chắc, lĩnh vực này đã đạt đến mức lắp ráp được những thiết bị phức tạp như: mô tô điện, rất có lợi về mặt kinh tế. Trong bài toán lắp ráp này, mục tiêu là tìm ra thứ tự lắp các phần của một vật. Nếu thứ tự lắp bị sai sẽ dẫn đến việc không thể gắn các phần khác vào hoặc phải tháo ra làm lại. Việc này khó khăn như việc tìm đường cho robot. Vì hàm trạng thái con có thể tốn nhiều chi phí. Bất kì thuật toán nào được ứng dụng cũng nên tránh việc xét hết toàn bộ không gian trạng thái. Ngoài ra thiết kế protein cũng là một bài toán thuộc loại này. Mục tiêu là tìm ra chuỗi amino axit ráp vào một protein 3 chiều với những tính chất phù hợp để chữa một số bệnh.

Tìm kiếm trên Internet

Những năm gần đây Internet ngày càng phát triển, nhu cầu tìm kiếm thông tin trên mạng để trả lời vô vàn câu hỏi, bổ sung thông tin ngày càng gia tăng. Từ đó cần có các robot tìm kiếm (phần mềm) trên mạng Internet để thực hiện công việc này. Đây là một ứng dụng hay của các kĩ thuật tìm kiếm, bởi ta có thể dễ dàng trừu tượng hóa Internet thành một đồ thị gồm các đỉnh (trang web) liên kết với nhau.

II. TÌM THEO CHIỀU RỘNG (Breadth-first search - BFS)

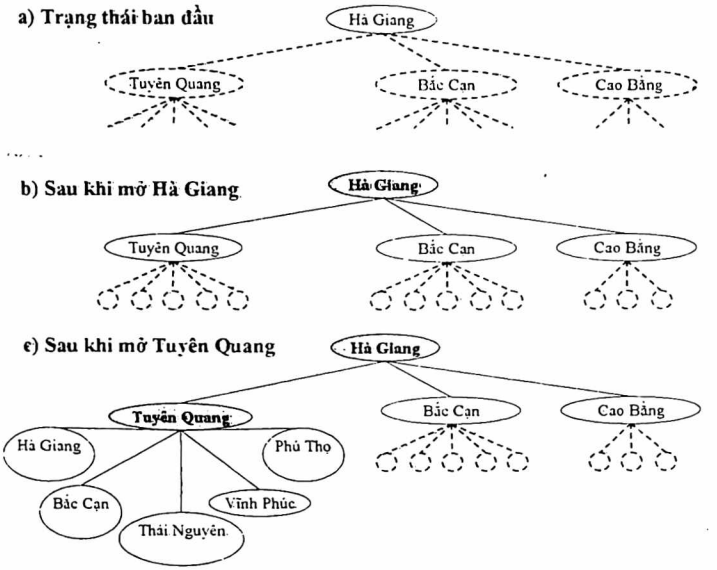
II.1 Ý tưởng: Tìm các đường đi trong một số bước nhất định.

Sau khi biểu diễn được bài toán, ta sẽ tiến hành giải quyết bài toán bằng cách tìm kiếm trong không gian trạng thái. Các phương pháp tìm kiếm thông thường dựa trên việc xây dựng **cây tìm kiếm** trong quá trình thực hiện. Cây tìm kiếm cũng được xây dựng từ trạng thái ban đầu và hàm trạng thái con của không gian trạng thái của bài toán tìm kiếm. Nút gốc của cây tìm kiếm sẽ là trạng thái ban đầu. Nút con của một nút trạng thái u là trạng thái được phát sinh từ u (nói cách khác là đỉnh kề đỉnh u trong đồ thị trạng thái). Trong trường hợp tổng quát chúng ta có một đồ thị tìm kiếm thay vì một cây tìm kiếm đơn thuần, vì có nhiều đường đi dẫn tới cùng một trạng thái. Ta sẽ xem xét đến trường hợp này trong phần IV.2.

Mỗi phương pháp tìm kiếm sẽ có một cách xây dựng cây tìm kiếm khác nhau. Hình 1.5 cho thấy một số bước đầu xây dựng cây tìm kiếm

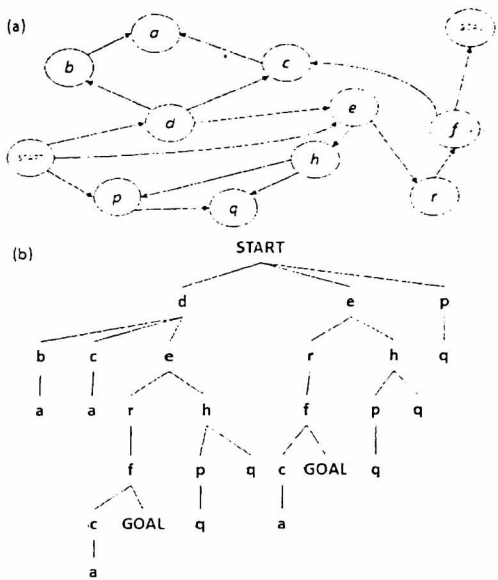
cho bài toán tìm đường đi du lịch từ *Hà Giang* đến *Hà Nội*. Xuất phát từ nút gốc là một **nút tìm kiếm** ứng với trạng thái ban đầu S. Bước đầu tiên là phải kiểm tra xem nút này có phải là đích không. Bước kiểm tra này rất quan trọng, mặc dù ta biết nút gốc là trạng thái ban đầu, nhưng nêu gặp phải yêu cầu tìm đường đi từ *Hà Giang* đến *Hà Giang* ta vẫn có thể giải được. Nút này không phải là trạng thái đích, ta sẽ xem xét tiếp những trạng thái khác bằng cách mở rộng trạng thái hiện tại, áp dụng hàm trạng thái cho trạng thái hiện tại để phát sinh ra một tập các trạng thái mới. Trong ví dụ minh họa có ba trạng thái mới được phát sinh từ nút gốc: Ở(Tuyên Quang), Ở(Bắc Cạn) và Ở(Cao Bằng). Tới bước này ta sẽ chọn một trong những trạng thái mới này để xem xét tiếp.

Giả sử chọn nút *Tuyên Quang*. Khi đó kiểm tra xem nút này có phải là trạng thái đích không (kết quả : không phải) và mở rộng nó thì thêm được các trạng thái mới Ở(Hà Giang), Ở(Phú Thọ), Ở(Bắc Cạn), Ở(Thái Nguyên) và Ở(Vĩnh Phúc). Ta có thể chọn một trong năm trạng thái mới này hoặc quay trở lại chọn *Bắc Cạn* hoặc *Cao Bằng* đều được. Tiếp tục công việc chọn, kiểm tra, mở rộng tới khi nào tìm thấy lời giải hoặc không thể mở rộng thêm trạng thái nào nữa thì dừng. Tùy thuộc vào chiến lược tìm kiếm mà sẽ có sự chọn lựa trạng thái khác nhau.



Hình 1.5 - *Một phần cây tìm kiếm cho bài toán tìm đường đi từ Hà Giang đến Hà Nội. Những nút đã mở rộng được tô đen, những nút mới phát sinh nhưng chưa được mở rộng được viền đen đậm, những nút chưa được phát sinh thể hiện ở dạng nét đứt mờ (dựa trên đồ thị không gian trạng thái).*

Cần phân biệt rõ sự khác nhau giữa không gian trạng thái với cây tìm kiếm. Trong bài toán tìm đường đi, số trạng thái trong không gian trạng thái là xác định (chẳng hạn là 18 ứng với số thành phố) nhưng số đường đi trong không gian trạng thái này lại là vô hạn dẫn đến số nút trên cây tìm kiếm cũng vô hạn. Ví dụ, các đường đi như Hà Giang-Tuyên Quang, Hà Giang-Tuyên Quang-Hà Giang, Hà Giang-Tuyên Quang-Hà Giang-Tuyên Quang chỉ là ba trong số dãy vô hạn những đường đi có thể có (*một thuật toán tìm kiếm tốt thường tránh những trường hợp lặp vô tận như vậy*). Hình 1.6 minh họa một không gian trạng thái và cây tìm kiếm tương ứng với không gian trạng thái này (đã loại bỏ những trường hợp lặp vô tận).

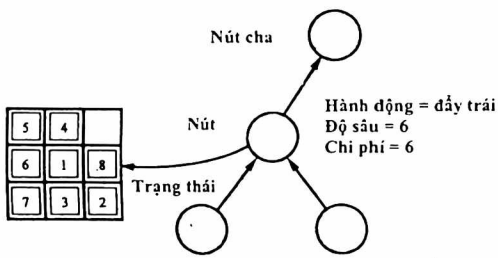


Hình 1.6 - (a) *Đồ thị không gian trạng thái của bài toán tìm đường đi trong đó START là trạng thái ban đầu, GOAL là trạng thái đích. (b) Cây tìm kiếm tương ứng với đồ thị không gian trạng thái.*

Có rất nhiều cách để biểu diễn một nút, ở đây ta xem nút là một cấu trúc dữ liệu gồm 5 thành phần sau:

- **Trạng thái:** Trạng thái tương ứng của nút trong không gian trạng thái.
- **Nút_Cha:** Nút cha đã phát sinh ra nút này trong cây tìm kiếm.
- **Hành động:** Hành động được thực hiện trên nút cha để phát sinh ra nút này.
- **Chi phí đường đi:** Chi phí của đường đi từ trạng thái ban đầu tới nút này được kí hiệu là g(n).
- **Độ sâu:** Số bước đi tính từ trạng thái ban đầu.

Ngoài ra, cũng cần phân biệt rõ giữa nút và trạng thái. Một nút là cấu trúc dữ liệu cơ bản tạo nên một cây tìm kiếm. Một trạng thái tương ứng với một tình trạng của thế giới (*thế giới của bài toán, không phải thế giới thực*). Do đó, nút thuộc về những đường đi cụ thể, trong khi trạng thái không như vậy. Vả lại, hai nút khác nhau có thể là của cùng một trạng thái, nếu trạng thái này được phát sinh bằng hai con đường khác nhau. Hình 1.7 minh họa cấu trúc dữ liệu của một nút trong cây tìm kiếm.



Hình 1.7 - *Nút là cấu trúc dữ liệu xây dựng nên cây tìm kiếm. Mỗi nút có một nút cha, một trạng thái và những thông tin khác. Chiều mũi tên đi từ nút con lên nút cha.*

Tập hợp những nút đã được phát sinh ra nhưng chưa được mở rộng gọi là **nhánh non**. Mỗi phần tử trong nhánh non là một nút lá tại thời điểm đang xét, được biểu diễn bởi những đường tròn trong hình 1.7.

Lúc này phương pháp tìm kiếm chính là một hàm thực hiện việc chọn nút tiếp theo để mở rộng trong tập hợp này, hoặc có thể xem như là một cách duyệt các nút trên cây tìm kiếm theo một thứ tự nhất định. Trên thực tế tập hợp nút này được cài đặt bằng cấu trúc dữ liệu hàng đợi tổng quát. Với các thao tác trên hàng đợi như sau:

```
function Tìm_kiếm_Cây(bài_toán,ngăn_chứa) return lời giải hoặc thất bại
 ngăn_chứa←Thêm(Tạo_Nút(Trạng_thái_đầu[bài_toán]), ngăn_chứa)
 loop do
        if Là_Rỗng(ngăn_chứa) then return thất bại
        nút←Lấy_phần_tử_đầu(ngăn_chứa)
        if Câu_hỏi_đích [bài_toán] trên Trạng_thái[nút] đúng
                then return Lời_giải(nút)
        ngăn_chứa←Thêm_Tất_cả(Mở_rộng(nút, bài_toán),ngăn_chứa)
function Mở_rộng(nút, bài_toán) return một tập các nút
 trạng_thái_con←khởi tạo tập rỗng
 for each (hành_động, kết_quả) in Con[bài_toán](Trạng_thái[nút]) do
        s ← một nút mới
        Trạng_thái[s] ←kết_quả
        Nút_Cha[s ] ←nút
        Hành_động[s] ←hành_động
        Chi_phí_đường_đi[s] ←Chi_phí_đường_đi[nút] + Chi_phí_bước
        (hành_động, Trạng_thái[nút], kết_quả)
        Độ_sâu[s] ←Độ_sâu [nút] + 1
        Thêm s vào trạng_thái_con
 return trạng_thái_con
```

***Hình 1.8** - Thuật toán tìm kiếm cây tổng quát. (Lưu ý: đối số ngăn_chứa phải là một hàng đợi rỗng và kiểu hàng đợi sẽ ảnh hưởng đến thứ tự tìm kiếm. Hàm Lời_giải trả về lời giải là một chuỗi các biến đổi có được từ con trỏ quay lui trở về nút gốc).*

- Tạo_Hàng_Đợi(phần_tử,...): Khởi tạo hàng với các *phần_tử*.
- Là_Rỗng(hàng_đợi): Trả về true nếu hàng đợi rỗng.
- Đầu(hàng_đợi): Trả về phần tử đầu tiên trong hàng đợi.
- Lấy_Phần_tử_Đầu(hàng_đợi): Trả về phần tử đầu tiên trong hàng đợi và loại nó ra khỏi hàng đợi.
- Thêm(phần_tử, hàng_đợi): Thêm một phần tử vào hàng đợi và trả về hàng đợi mới.
- Thêm_Tất_cả(các_phần_tử, hàng_đợi): Thêm một loạt các phần tử vào hàng đợi và trả về hàng đợi mới.

Tùy vào kiểu hàng đợi mà thao tác thêm vào và lấy ra có sự khác biệt. Trong giáo trình này đề cập đến ba kiểu hàng đợi:

- Hàng đợi First-In-First-Out (FIFO): Thường gọi tắt là hàng đợi.
- Hàng đợi Last-In-First-Out (LIFO): Còn gọi là ngăn xếp (Stack).
- Hàng đợi ưu tiên (Priority Queue).

Với những khái niệm trên, ta có thể viết thuật toán tìm kiếm tổng quát dựa trên cây tìm kiếm như trong Hình 1.8.

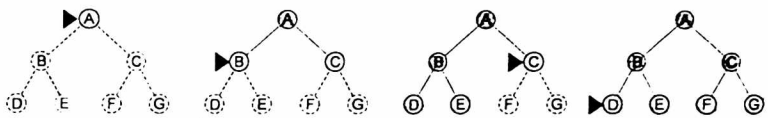
II.2 Con trỏ quay lui

Trong quá trình xây dựng cây tìm kiếm, mỗi nút đều chứa thông tin về trạng thái tương ứng, bên cạnh đó còn ghi nhận cả trạng thái trước đó (nút cha của nó). Với mục tiêu là tìm đường đi từ trạng thái ban đầu tới trạng thái đích nên cần lưu vết lại, để từ đó có thể phát sinh lời giải (con đường để đi tới đích). Điều này rất quan trọng vì nếu chỉ quan tâm đến việc đã tới đích hay chưa, mà không ghi nhớ lại trước đó đã đi như thế nào để tới đích, thì như vào được trong mê cung mà không ra lại được. Việc ghi nhận lại trạng thái trước đó được gọi là con trỏ quay lui. Con trỏ quay lui được lưu trong một nút bởi ***Nút_Cha***, trong đó ***Nút_Cha***(v) = u, nghĩa là u là nút cha của v.

II.3 Thuật toán tìm kiếm theo chiều rộng (BFS)

Tìm kiếm theo chiều rộng ứng với việc xây dựng cây tìm kiếm theo chiều rộng, nút gốc được mở rộng đầu tiên, sau đó tất cả những nút con có được từ nút gốc sẽ được mở rộng, tiếp theo và sau đó là nút con của chúng và cứ thế tiếp tục. Tổng quát, ta có thể thực hiện như sau: tất cả các nút ở độ sâu là d trong cây tìm kiếm sẽ được mở rộng trước những nút ở độ sâu d+1. Cách làm này xuất phát từ ý tưởng là sẽ có một hoặc nhiều đường đi tới đích qua n bước. Do n bước ta chưa xác định được nên sẽ phải xét hết các đường đi có thể lần lượt từ n = 1, 2, 3... tới khi nào gặp đích thì dừng. Với cách làm này chúng ta phủ toàn bộ các nhánh có thể của cây tìm kiếm với độ cao nhỏ hơn n, giống như việc ta loang rộng trên cây tìm kiếm nên chiến lược này được gọi là *tìm kiếm theo chiều rộng*.

Thuật toán tìm theo chiều rộng có thể cài đặt dựa trên thuật toán tìm kiếm tổng quát, sử dụng hàm Tìm_kiếm_Cây với tham số *ngăn_chứa* truyền vào là một hàng đợi FIFO theo cách: nút xét đến trước sẽ được mở rộng trước. Nói cách khác, việc gọi hàm Tìm_kiếm_Cây(*bài_toán*, Hàng_đợi_FIFO()) sẽ cho kết quả tìm kiếm theo chiều rộng. Hàng đợi FIFO đặt những nút mới phát sinh vào cuối hàng đợi, tương đương với việc mở rộng những nút ở gần trước khi mở rộng những nút sâu hơn. Hình 1.9 minh họa thứ tự duyệt của thuật toán trên cây nhị phân đơn giản.



Hình 1.9 - *Tìm kiếm theo chiều rộng trên cây nhị phân đơn giản. Tại mỗi bước, đầu mũi tên chỉ ngay vào nút sẽ được mở rộng tiếp theo.*

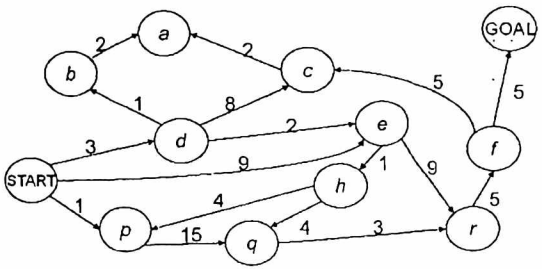
III. TÌM KIẾM CHI PHÍ ĐỒNG NHẤT (Uniform-cost Search - UCS)

Thuật toán BFS chỉ tìm ra đường đi có số bước biến đổi ít nhất. Nó không tìm thấy đường đi có chi phí nhỏ nhất. Đối với những bài toán tìm kiếm mà các bước biến đổi có chi phí, nghĩa là đồ thị trạng thái là đồ thị có trọng số, thì thuật toán BFS cho ra lời giải không phù hợp.

Hình 1.10 là đồ thị biểu diễn cho bài toán tìm kiếm có chi phí. Đây là mở rộng của bài toán biểu diễn bởi Hình 1.6 (a) với sự bổ sung chi phí cho mỗi đường đi.

Thuật toán tìm kiếm theo chiều rộng sẽ tốt trong trường hợp các bước biến đổi có chi phí như nhau, bởi vì nó luôn mở rộng nút gần nhất. Một mở rộng đơn giản cho thuật toán tìm kiếm theo chiều rộng để tìm lời giải khi có hàm chi phí chuyển đổi bất kì, Đó là thay vì mở rộng nút gần nhất, thuật toán tìm kiếm chi phí đồng nhất sẽ mở rộng nút n có chi phí đường đi tới nó thấp nhất. Thuật toán tìm kiếm chi phí đồng nhất tổng quát hơn thuật toán tìm kiếm theo chiều rộng, trong trường

hợp đặc biệt (các bước biến đổi có chi phí như nhau) hai thuật toán này là như nhau.



Hình 1.10 - *Bài toán tìm kiếm có chi phí.*

Tương tự như tìm kiếm theo chiều rộng, thuật toán tìm kiếm chi phí đồng nhất thường được cài đặt dựa trên thuật toán tìm kiếm tổng quát, sử dụng hàm Tìm_kiếm_Cây với tham số *ngăn_chứa* truyền vào là một hàng đợi ưu tiên theo chi phí chuyển đổi. Nói cách khác, việc gọi hàm Tìm_kiếm_Cây(*bài_toán*, Hàng_đợi_Ưu_Tiên()) sẽ cho kết quả tìm kiếm chi phí đồng nhất. Hàng đợi ưu tiên được trình bày trong phần sau.

III.1 Hàng đợi ưu tiên

Hàng đợi ưu tiên là một cấu trúc dữ liệu trong đó ta có thể thêm và lấy ra các phần tử dựa trên độ ưu tiên của mỗi phần tử. Mỗi phần tử trong hàng đợi thường có kèm theo một giá trị dùng để xác định độ ưu tiên của phần tử đó trong hàng đợi.

Hàng đợi ưu tiên có hai toán tử (hoặc thao tác) cơ bản:

- **Thêm**: Đưa một phần tử vào hàng đợi.
- **Lấy**: Lấy ra một phần tử trong hàng đợi có độ ưu tiên cao nhất.

Đối với thuật toán tìm kiếm chi phí đồng nhất, độ ưu tiên của một phần tử (trạng thái) chính là chi phí đường đi tới trạng thái đó, trạng thái nào có chi phí đường đi thấp hơn thì độ ưu tiên sẽ cao hơn và trạng thái đó sẽ được lấy ra khỏi hàng đợi trước.

Ta có thể cài đặt hàng đợi ưu tiên theo nhiều phương pháp khác nhau chẳng hạn ta có thể dùng:

- Mảng tĩnh.
- Danh sách liên kết: Thao tác **thêm** tại đầu (hoặc cuối) danh sách có chi phí là O(1), khi **lấy** ra tiến hành duyệt danh sách để tìm phần tử có độ ưu tiên cao nhất cần thời gian O(*N*). Hoặc, giữ danh sách luôn được sắp xếp theo độ ưu tiên, như vậy thao tác **thêm** tốn chi phí O(*N*) nhưng thao tác **lấy** chi cần O(1).
- Cây nhị phân tìm kiếm: Cả hai thao tác **thêm** và **lấy** đều có độ phức tạp là O(log*N*). Đây là cách cài đặt tối ưu.

Với N là số phần tử trong hàng đợi ưu tiên.

III.2 Hàm chi phí

Tuy ở trên có đề cập hàm chi phí chuyển đổi bất kì, nhưng thuật toán chi phí đồng nhất cần điều kiện ràng buộc đối với hàm chi phí để tránh trường hợp lặp vô hạn. Chi phí đường đi tại mỗi bước biến đổi phải lớn hơn hoặc bằng một hằng số *c* dương.

III.3 Thuật toán và điều kiện dừng.

Thuật toán tìm kiếm chi phí đồng nhất có thể được viết lại chi tiết như sau:

```
function Tìm_kiếm_UCS (bài_toán, ngăn_chứa) return lời giải hoặc thất bại.
ngăn_chứa←Tạo_Hàng_Đợi_Rỗng()
ngăn_chứa←Thêm(TẠO_NÚT(Trạng_Thái_Đầu[bài_toán]), ngăn_chứa)
loop do
        if Là_Rỗng(ngăn_chứa) then return thất bại.
        nút←Lấy_Chi_phí_Nhỏ_nhất(ngăn_chứa)
        if Kiểm_tra_Câu_hỏi_đích[bài_toán] trên Trạng_thái[nút] đúng.
                then return Lời_giải(nút).
        lg ←Mở(nút, bài_toán)                    //lg tập các nút con mới.
        ngăn_chứa←Thêm_Tất_cả(lg, ngăn_chứa)
```

Hình 1.11 - *Thuật toán tìm kiếm UCS.*

Lưu ý với mô tả thuật toán tìm kiếm như trên, thuật toán sẽ dừng khi *đích được lấy ra khỏi hàng đợi ưu tiên*. Dòng lệnh kiểm tra là đích được thực hiện ngay sau thao tác lấy một phần tử ra khỏi hàng đợi.

Nếu ta thực hiện kiểm tra đích ngay sau thao tác mở rộng một nút thì có thể sẽ bỏ qua đường đi ngắn nhất.

Trong thuật toán tìm kiếm UCS, thao tác chính trong mỗi vòng lặp bao gồm *lấy phần tử có chi phí thấp nhất* ra khỏi hàng đợi ưu tiên và *thêm các trạng thái con của phần tử vừa lấy*. Quá trình thực hiện thuật toán này có thể được biểu diễn bằng các trạng thái của hàng đợi ưu tiên qua các vòng lặp. Phần dưới đây mô tả các bước thực hiện cua tìm kiếm UCS đối với ví dụ trong Hình 1.10.

1. PQ = {(Start, 0)}.
2. PQ = {(p, 1), (d, 3), (e, 9)}.
3. PQ = {(d, 3), (e, 9), (q, 16)}.
4. PQ = {(b, 4), (e, 5), (c, 11), (q, 16)}.
5. PQ = {(e, 5), (a, 6), (c, 11), (q, 16)}.
6. PQ = {(a, 6), (h, 6), (c, 11), (r, 14), (q, 16)}.
7. PQ = {(h, 6), (c, 11), (r, 14), (q, 16)}.
8. PQ = {(q, 10), (c, 11), (r, 14)}.
9. PQ = {(c, 11), (r, 13)}.
10. PQ = {(r, 13)}.
11. PQ = {(f, 18)}.
12. PQ = {Goal, 23)}.

Vậy, đường đi ngắn nhất từ Start đến Goal là: Start → d → c → h → q → r→ f → Goal với chi phí là 23.

III.4 Đánh giá thuật toán tìm kiếm

Một thuật toán tìm kiếm để giải quyết bài toán có thể cho ra một lời giải hoặc thất bại (kết thúc nhưng không tìm thấy lời giải hoặc lặp vô hạn không kết thúc). Ta sẽ đánh giá thuật toán tìm kiếm dựa trên bốn yếu tố sau:

- **Tính đầy đủ:** Thuật toán có bảo đảm tìm thấy lời giải nếu có hay không?
- **Tính tối ưu:** Thuật toán có bảo đảm tìm thấy lời giải tối ưu không? (Sẽ tìm thấy đường đi có chi phí nhỏ nhất?)
- **Độ phức tạp về thời gian:** Mất bao lâu để tìm ra lời giải.

- **Độ phức tạp về không gian:** Cần sử dụng bao nhiêu bộ nhớ trong quá trình tìm kiếm.

Độ phức tạp về thời gian và không gian thường được xem như các khía cạnh của việc đo lường độ phức tạp của bài toán. Theo lý thuyết trong khoa học máy tính, thước đo thông thường là kích thước của đồ thị không gian trạng thái, bởi vì đồ thị này được xem là một cấu trúc dữ liệu tường minh làm đầu vào cho thuật toán tìm kiếm (Ví dụ như bản đồ du lịch). Trong trí tuệ nhân tạo, thì đồ thị này được thể hiện ngầm bởi trạng thái ban đầu và hàm trạng thái con. Độ phức tạp được thể hiện bằng ba đại lượng sau:

- b: Hệ số phân nhánh trung bình (số con trung bình hoặc tối đa) (b>1).
- D: Độ dài đường đi từ nút gốc đến nút đích với số bước (chi phí) ít nhất.
- m: Độ dài đường đi dài nhất từ nút gốc đến một nút bất kì.

Độ phức tạp về thời gian được tính bằng số lượng trạng thái con được phát sinh trong quá trình tìm kiếm, về không gian được tính bằng số trạng thái lớn nhất được lưu trong bộ nhớ.

Để đánh giá hiệu quả của một thuật toán tìm kiếm, có thể chỉ cần xem xét đến chi phí tìm kiếm, thường liên quan tới độ phức tạp về thời gian. Ngoài ra, có thể tính chi phí tổng cộng trong đó xem xét thêm về việc sử dụng bộ nhớ bao gồm chi phí tìm kiếm và chi phí đường đi của lời giải tìm ra được. Ví dụ, trong bài toán tìm đường đi từ thành phố *Hà Giang* tới *Hà Nội* thì chi phí tìm kiếm là thời gian cần cho việc tìm kiếm (ms) và chi phí của lời giải là tổng chiều dài đường đi (km). Như vậy, để tính chi phí tổng cộng ta phải cộng hai đại lượng có đơn vị là km và mili giây với nhau. Không có công thức chuyển đổi giữa hai đơn vị này, nhưng ta có thể tạm chấp nhận chuyển km sang mili giây bằng cách xấp xỉ theo tốc độ xe chạy trung bình (bài toán này quan tâm về thời gian).

Sử dụng bốn tiêu chuẩn được đề cập ở trên, ta sẽ lần lượt đánh giá thuật toán tìm kiếm theo chiều rộng và thuật toán tìm kiếm chi phí đồng nhất.

Đối với thuật toán tìm kiếm theo chiều rộng, dễ thấy rằng thuật toán thỏa tính đầy đủ. Nếu nút đích gần nhất nằm ở một độ sâu d xác định (trên cây tìm kiếm), thuật toán chắc chắn sẽ tìm thấy nút này sau khi mở rộng hết tất cả những nút ở độ sâu gần hơn (với điều kiện b hữu hạn). Nút đích gần nhất không nhất thiết là lựa chọn tối ưu, nhưng thuật toán tìm kiếm theo chiều rộng vẫn tối ưu nếu chi phí chuyển đổi là một hàm không giảm theo độ sâu của nút (chẳng hạn trong trường hợp tất cả các biến đổi đều có cùng chi phí). Đến đây, ta thấy thuật toán tìm kiếm theo chiều rộng khá tốt khi thỏa cả hai tiêu chuẩn đầu. Hai tiêu chuẩn sau sẽ cho thấy tại sao thuật toán tìm kiếm theo chiều rộng không là lựa chọn hàng đầu. Giả sử mọi trạng thái có thể mở rộng được tối đa b nút. Gốc của cây tìm kiếm sẽ tạo ra b nút ở mức đầu tiên, mỗi nút con mới đó sẽ tạo ra thêm b nút con nữa, tổng cộng sẽ có b^2 nút ở độ sâu thứ hai, cứ thế tiếp tục. Giả sử rằng đường đi tới đích có chiều dài là d. Trong trường hợp xấu nhất sẽ phải mở rộng các nút ở độ sau d (trừ nút đích không mở rộng). Khi đó tổng số nút được phát sinh ra sẽ là:

$$b^1 + b^2 + b^3 + \ldots + b^d + (b^{d+1} - b^1) = O(b^{d+1})$$

Các nút được phát sinh ra đều được lưu trong bộ nhớ, do đó độ phức tạp về mặt không gian giống với độ phức tạp về mặt thời gian và bằng $O(b^d)$.

Độ phức tạp là một hàm mũ, như vậy thời gian cũng như không gian sử dụng sẽ tăng rất nhanh theo độ sâu. Điều này được thể hiện qua Bảng 1-1 cho thấy thời gian và dung lượng bộ nhớ cần cho thuật toán tìm kiếm theo chiều rộng với $b = 10$ và theo giá trị độ sâu d tăng dần. Giả sử có thể phát sinh 10.000 nút trong vòng 1 giây và mỗi nút cần 1.000 byte để lưu trữ.

Bảng 1-1. *Thể hiện thời gian chạy và bộ nhớ sử dụng của tìm kiếm theo chiều rộng. Với giả thiết là b = 10; 10.000 nút/giây, 1.000 byte/nút.*

Độ sâu	Số nút	Thời gian	Bộ nhớ
2	1100	11 giây	1MB
4	111100	11 giây	106 MB
6	10^7	19 phút	10 GB
8	10^9	31 giờ	1 TB
10	10^{11}	129 ngày	101 TB
12	10^{13}	35 năm	10 PB
14	10^{15}	3523năm	1 EB

Từ bảng trên, sẽ thấy được hai điều. Đầu tiên là trong tìm kiếm theo chiều rộng, yêu cầu về bộ nhớ là vấn đề lớn hơn so với thời gian chạy. Người ta có thể ngồi chờ 31 giờ để tìm ra lời giải ở độ sâu là 8 được, nhưng không có máy tính nào bộ nhớ lên đến hàng Terabyte để sử dụng. Điều thứ hai là thời gian vẫn còn là một nhân tố ảnh hưởng khá lớn. Nếu bài toán cần giải có lời giải ở độ sâu là 12 thì sẽ cần đến 35 năm để tìm ra nó bằng thuật toán tìm kiếm theo chiều rộng.

Thuật toán tìm kiếm chi phí đồng nhất không quan tâm đến số bước của đường đi, mà chỉ quan tâm đến chi phí tổng cộng. Do đó, thuật toán có thể bị lặp vô hạn, nếu gặp một nút có chi phí chuyển đổi tới cùng trạng thái đó bằng 0 (ví dụ ứng hành động không làm gì cả). Chúng ta có thể đảm bảo được tính đầy đủ với điều kiện chi phí tại mỗi bước biến đổi lớn hơn hoặc bằng một hằng số c dương đủ nhỏ. Điều kiện này cũng đảm bảo tính tối ưu. Điều này đồng nghĩa với hàm chi phí sẽ tăng theo đường đi, việc mở rộng thêm nút sẽ làm tăng chi phí của đường đi. Do đó, khi bắt gặp nút đích đầu tiên thì đó cũng chính là lời giải tối ưu.

Thuật toán tìm kiếm chi phí đồng nhất chủ yếu dựa trên chi phí của đường đi nên độ phức tạp khó xác định bởi giá trị của b và d (độ sâu). Thay vào đó, gọi C^* là chi phí của lời giải tối ưu và giả sử rằng chi phí chuyển đổi ít nhất là e. Như vậy độ phức tạp về thời gian và không gian của thuật toán là $O(b^{1+\lfloor C^*/e \rfloor})$ và có thể lớn hơn b^d nhiều. Bởi vì,

tìm kiếm chi phí đồng nhất có thể và thông thường sẽ mở rộng hướng tìm kiếm theo những nhánh mà ban đầu có chi phí thấp trước khi đi vào con đường đúng mà ban đầu có chi phí cao (tổng chi phí vẫn thấp hơn). Khi chi phí chuyển đổi đều bằng nhau thì $b^{1+\lfloor C^*/\varepsilon \rfloor}$ sẽ chỉ còn là b^d.

Bảng 1-2. *Đánh giá thuật toán BFS và UCS. *Nếu chi phí chuyển đổi bằng nhau.*

Thuật toán	Đủ	Tối ưu	Thời gian	Không gian
BFS	Có	Có*	$O(b^{d+1})$	$O(b^{d-1})$
UCS	Có	Có	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$

IV. TÌM THEO CHIỀU SÂU (Depth-first search - DFS)

IV.1 Tìm kiếm theo chiều sâu – Một thuật toán thay thế.

Tìm kiếm theo chiều sâu luôn mở rộng nút nằm sâu nhất trong nhánh hiện tại đang xét trên cây tìm kiếm. Quá trình được thể hiện trong Hình 1.12. Thuật toán tiến hành tìm kiếm ở mức sâu nhất của cây tìm kiếm, tại đó các nút không có nút con nào. Với những nút đã được mở rộng sẽ được loại khỏi nhánh tìm kiếm, khi đó công việc tìm kiếm sẽ quay trở lại nút gần nhất có các nút con chưa được xét đến.

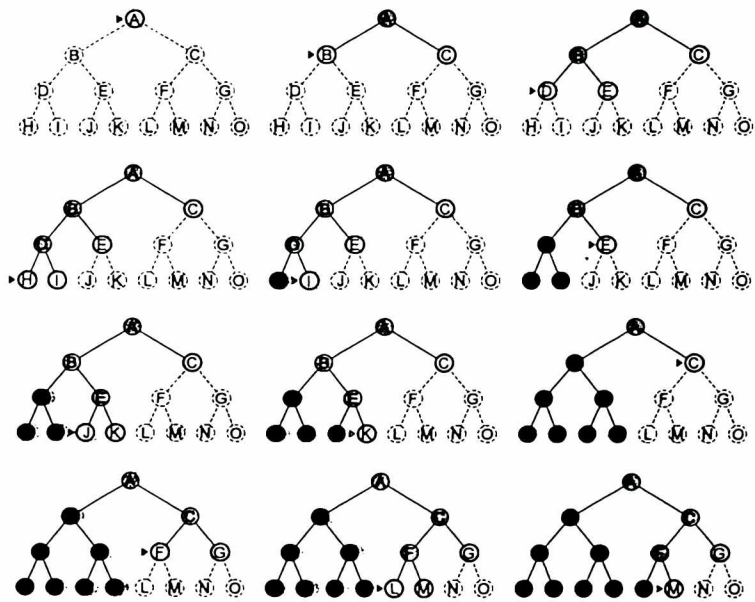
```
function Tìm_kiếm_Chiều_sâu(bài_toán) return lời giải hoặc thất bại.
    return DFS_Đệ_qui(TẠO_NÚT(TThái_Đầu[bài_toán]), bài_toán.)

function DFS_Đệ_qui(nút, bài_toán) return lời giải hoặc thất bại.
    if Kiểm_tra_Câu_hỏi_đích[bài_toán] trên Trạng_thái[nút] đúng.
        then return Lời_giải(nút).
    else for each con in Mở_rộng(bài_toán) do
        kết_quả ← DFS_Đệ_qui(con, bài_toán).
        if kết_quả ≠ thất bại then return kết_quả.
    return thất bại.
```

Hình 1.12 - *Thuật toán tìm kiếm Độ_sâu-First-Search.*

Thuật toán tìm kiếm theo chiều sâu luôn mở rộng nút sâu mãi theo một hướng, chỉ khi nào không mở rộng được nữa thì chuyển sang hướng khác. Thuật toán có thể cài đặt dựa trên thuật toán tìm kiếm

tổng quát, sử dụng hàm Tìm_kiếm_Chiều_sâu với tham số *ngăn_chứa* truyền vào là một hàng đợi LIFO còn gọi là ngăn xếp. Theo nguyên tắc này thì những nút được mở rộng sau (có độ sâu lớn hơn) sẽ được lấy ra trước. Một cách cài đặt khác, thường được sử dụng cho thuật toán tìm kiếm theo chiều sâu, đó là sử dụng một hàm đệ qui gọi lại chính nó lần lượt cho mỗi nút trên cây. Cách cài đặt này sẽ được minh họa sau đây.



Hình 1.13 - *Tìm kiếm theo chiều sâu trên cây nhị phân. Những nút đã được mở rộng và không mở rộng thêm được nút con nào, có thể được xóa khỏi bộ nhớ. Những nút này được tô màu đen đậm. Với giả thiết là nút ở độ sâu thứ 3 không có nút con nào và M là nút đích duy nhất.*

Thuật toán tìm kiếm theo chiều sâu sử dụng bộ nhớ khá tối ưu. Qua Hình 1.13 ta thấy thuật toán chỉ lưu trữ những nút (từ nút gốc đến nút

lá) trên nhánh đường đi đang được xem xét đến, cùng với những nút lân cận chưa được mở rộng trên đường đi. Một nút sẽ được xóa khỏi bộ nhớ khi nó đã được mở rộng và các nút con cháu của nó đã được xét hết. Trong không gian trạng thái, với số nút con tối đa là b và độ sâu tối đa là m, DFS chỉ cần lưu tối đa $b \times m + 1$ nút. Ở mỗi độ sâu, mở rộng được tối đa là b nút, với độ sâu m thì trong bộ nhớ sẽ lưu $b \times m + 1$ nút (tính luôn nút gốc). Trong khi đó thì tìm kiếm theo chiều rộng cần đến b^d nút trong trường hợp nút đích ở độ sâu là d.

Độ phức tạp về mặt thời gian của thuật toán tìm kiếm theo chiều sâu là $O(b \times m)$. Đối với những vấn đề có nhiều lời giải, thì tìm kiếm theo chiều sâu thực sự nhanh hơn tìm kiếm theo chiều rộng, bởi vì nó có nhiều cơ hội tìm ra lời giải sau khi mở rộng một số lượng nhỏ các nút trong toàn không gian. Trong khi tìm kiếm theo chiều rộng sẽ phải xét hết tất cả đường đi có chiều dài $d-1$có thể trước khi xét đến các đường đi có chiều dài là d. Với cùng một giả thiết, thuật toán tìm kiếm theo chiều sâu chỉ cần 118KB thay vì 10PB ($\sim 10^{16}$ byte) như trong tìm kiếm theo chiều rộng, khoảng chênh lệch về không gian bộ nhớ sử dụng lên đến 10 tỉ lần.

Một biến thể của thuật toán tìm kiếm theo chiều sâu gọi là tìm kiếm quay lui, thuật toán này sử dụng ít bộ nhớ hơn. Trong tìm kiếm quay lui, tại một thời điểm chỉ có một nút con được sinh ra. Mỗi nút con này sẽ lưu giữ thông tin về nút cha của nó. Với cách này thì chỉ cần $O(m)$ không gian bộ nhớ thay vì là $O(bm)$. Ngoài ra, thuật toán này còn sử dụng một thủ thuật tiết kiệm bộ nhớ (có thể nói là tiết kiệm thời gian) khác nữa. Với ý tưởng khi phát sinh một nút con mới ta sẽ trực tiếp thay đổi thông tin về trạng thái hiện tại thay vì sao chép lại. Cách này làm giảm yêu cầu về bộ nhớ xuống còn lưu trữ thông tin một trạng thái và $O(m)$ thao tác. Để có thể làm được điều này, phải có khả năng phục hồi lại những thay đổi trước đó khi ta quay lại phát sinh tiếp trạng thái mới. Đối với những bài toán có các trạng thái phức tạp, chẳng hạn lắp ráp tự động, những kĩ thuật này rất quan trọng.

IV.2 Tránh lặp vô hạn – Các chiến lược kiểm tra

Nhược điểm của tìm kiếm theo chiều sâu là nó có thể bị kẹt (xuống sâu mãi mà không thoát ra được) nếu đi không đúng đường, trong khi

nếu chọn đường khác thì lời giải lại nằm rất gần nút gốc của cây tìm kiếm. Ví dụ trong Hình 1.13 thuật toán sẽ mở rộng toàn bộ cây con bên trái trước ngay cả khi C là nút đích. Nếu J cũng là nút đích thì thuật toán sẽ tìm thấy và cho ra lời giải nhưng lại không phải lại lời giải tối ưu. Do đó, tìm kiếm theo chiều sâu không tối ưu. Nếu cây con bên trái có độ sâu không giới hạn và cũng không chứa nút đích, tìm kiếm theo chiều sâu sẽ lặp vô hạn. Do đó, tìm kiếm theo chiều sâu không đầy đủ. Trong trường hợp xấu nhất, tìm kiếm theo chiều sâu sẽ phát sinh tất cả $O(b^m)$ nút trong cây tìm kiếm, với m là độ sâu lớn nhất của một nút bất kì. Nên nhớ là m lớn hơn d rất nhiều và có thể vô hạn nếu cây không có giới hạn.

Nhiều bài toán có lời giải nằm rất sâu hoặc độ sâu không xác định, nên tìm kiếm theo chiều sâu sẽ không bao giờ chuyển sang hướng khác được, nếu không may rơi vào một hướng không có lời giải. Việc tìm kiếm sẽ cứ đi xuống sâu mãi một hướng, mặc dù ở hướng khác có lời giải.

Nhiều trường hợp tìm kiếm theo chiều sâu sẽ bị lặp vô tận và không bao giờ tìm ra lời giải. Điều này có nghĩa là tìm kiếm theo chiều sâu có khả năng không tìm ra lời giải cũng như là lời giải tối ưu. Bởi vậy, tìm kiếm theo chiều sâu nên tránh sử dụng cho những vấn đề mà cây tìm kiếm lớn hoặc độ sâu tối đa không xác định.

Ngoài ra, các trạng thái có thể bị lặp lại trên cây tìm kiếm cũng khiến cho thuật toán tìm kiếm bị lặp vô tận. Trên cây tìm kiếm, các trạng thái trùng lặp lại tương ứng với những trạng thái có nhiều con đường cùng dẫn tới nó từ trạng thái ban đầu trong đồ thị không gian trạng thái. Nếu đồ thị này có chu trình thì cây tìm kiếm sẽ vô hạn, với các nhánh có một số nút được lặp lại vô hạn lần. Để tránh sự lặp lại vô tận như vậy, ta có thể áp dụng một trong các chiến lược sau:

- Khi mở rộng nút n sẽ không phát sinh các nút mới trùng với nút cha của n.
- Khi mở rộng nút n sẽ không phát sinh các nút mới trùng với bất kì một nút nào, nằm trên đường đi từ nút gốc tới n.
- Không mở rộng các nút đã được mở rộng trước đó.

Hai chiến lược đầu không tốn nhiều không gian bộ nhớ, tuy nhiên lại không tránh được hết các trường hợp lặp. Chiến lược thứ ba đòi hỏi phải lưu tất cả các trạng thái đã mở rộng trong bộ nhớ. Vấn đề này sẽ được đề cập chi tiết hơn trong phần V.

Tìm kiếm với độ sâu giới hạn

Tìm kiếm với độ sâu giới hạn khắc phục được nhược điểm cây tìm kiếm không giới hạn của tìm kiếm theo chiều sâu, bằng cách giới hạn ở một độ sâu cho trước. Những nút nằm ở độ sâu lớn hơn sẽ không được xét đến. Tuy nhiên cải tiến này lại nảy sinh vấn đề đối với tính đầy đủ và tối ưu của thuật toán tìm kiếm. Nếu chọn độ sâu giới hạn $l < d$ đồng nghĩa với việc nút đích gần nhất nằm dưới độ sâu giới hạn (có thể xảy ra khi chưa biết d) thì thuật toán sẽ không tìm ra lời giải. Thuật toán cũng không tối ưu khi chọn $l > d$. Độ phức tạp về mặt thời gian và không gian là tương tự nhau đều là $O(b^l)$.

Phần khó nhất của tìm kiếm với độ sâu giới hạn là chọn độ sâu thích hợp. Hầu hết mọi vấn đề, ta sẽ không biết được độ sâu giới hạn tốt nhất cho tới khi vấn đề đã được giải quyết. Ngoài ra, ta cũng có thể dựa vào những hiểu biết về bài toán để giới hạn lại phạm vi tìm kiếm trong không gian trạng thái.

Có thể cài đặt thuật toán tìm kiếm, dựa trên thuật toán tìm kiếm tổng quát trên cây với một vài điều chỉnh cho phù hợp hoặc dựa trên dạng đệ qui của thuật toán tìm kiếm theo chiều sâu. Cho trước một độ sâu nào đó, rồi tiến hành tìm kiếm chiều rộng ứng với mức sâu đó. Đoạn mã giả của thuật toán tìm kiếm với độ sâu giới hạn được trình bày trong Hình 1.14.

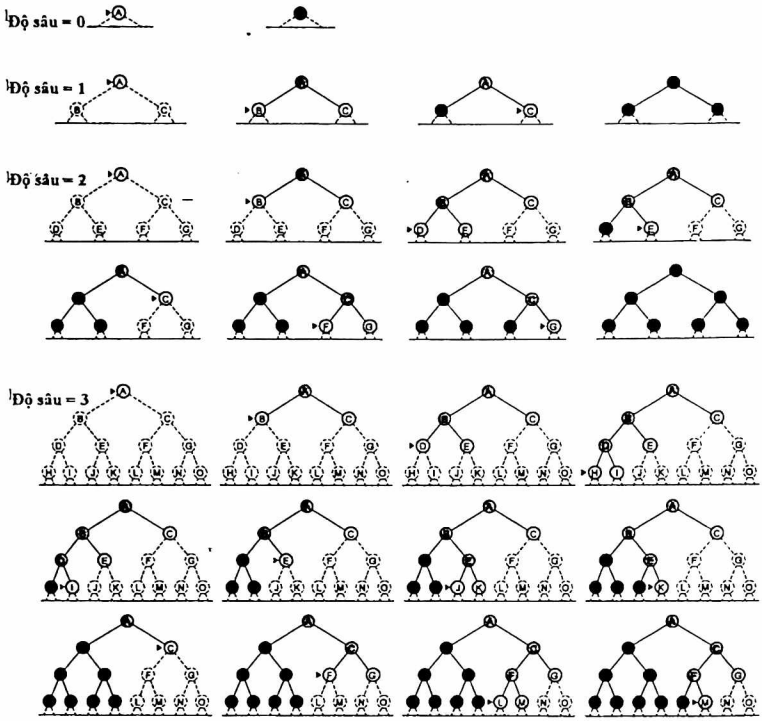
```
function Tìm_kiếm_Độ_sâu_Giới hạn(bài_toán, giới_hạn) return lời giải hoặc
                                                              thất bại
    return TĐG_Đệ_qui(Tạo_nút(Trạng_thái_đầu[bài_toán]), bài_toán,
                       giới_hạn)
function TĐG_Đệ_qui (nút, bài_toán, giới_hạn) return lời giải hoặc thất
                                                              bại/dừng
    có_tia_nhánh ← false
    if Kiểm_tra_Câu_hỏi_đích[bài_toán] trên Trạng_thái[nút] đúng
        then return Lời_giải(nút)
    else if Độ_sâu[nút] = giới_hạn then return dừng
    else for each nút_con in Mở_rộng(bài_toán) do
        kết_quả ← TĐG_Đệ_qui(nút_con, bài_toán)
        if kết_quả = dừng then có_tia_nhánh ← true
        else if kết_quả ≠ thất bại then return kết_quả
    if có_tia_nhánh then return dừng return thất bại.
```

Hình 1.14 - *Thuật toán Tìm kiếm Độ sâu Giới hạn.*

Lưu ý, hàm đệ qui của tìm kiếm với độ sâu giới hạn trong trường hợp không tìm ra lời giải sẽ trả về một trong hai dạng: thất bại khi không tìm thấy lời giải và giá trị dừng cho biết không có lời giải nào trong độ sâu giới hạn.

Tìm lời giải tối ưu – Tìm kiếm lặp sâu dần (Iterative Deepening Search – IDS)

Tìm kiếm *lặp sâu dần* (hay còn gọi tìm kiếm với *độ sâu tăng dần*) là một phương pháp cải tiến vấn đề chọn độ sâu nào tốt nhất này, đơn giản bằng cách lần lượt thử hết các độ sâu có thể. Đầu tiên là 0, sau đó là 1 rồi 2... Tìm kiếm với chiều sâu tăng dần đã kết hợp được những ưu điểm của tìm kiếm theo chiều rộng và chiều sâu. Nó vừa tối ưu (khi chi phí không giảm theo độ sâu), vừa đầy đủ (khi số nhánh con tối đa *b* hữu hạn), giống như tìm kiếm theo chiều rộng, nhưng lại có được sự tối ưu về bộ nhớ $O(bd)$ của tìm kiếm theo chiều sâu. Thứ tự



Hình 1.15 - *Bốn bước lặp của thuật toán lặp sâu dần trên cây nhị phân.*

hạn. Hình 1.15 cho thấy bốn bước lặp đầu của hàm Tìm_kiếm_Lặp_Sâu_dần trên một cây tìm kiếm nhị phân và lời giải được tìm ra ở bước lặp thứ tư.

```
function Tìm_kiếm_Lặp_Sâu_dần(bài_toán) return lời giải hoặc thất bại
    for độ_sâu ← 0 to ∞ do
        kết_quả ←Tìm_kiếm_Độ_sâu_Giới hạn(bài_toán, độ_sâu)
        if kết_quả ≠ dừng then return kết_quả
```

Hình 1.16 -*Thuật toán Tìm kiếm lặp sâu dần.*

Tìm kiếm với chiều sâu tăng dần có vẻ lãng phí, do có nhiều trạng thái được phát sinh nhiều lần. Nhưng số lượng trạng thái thực sự không quá lớn. Bởi vì, trong cây tìm kiếm với hệ số phân nhánh gần như nhau ở mỗi mức sâu thì các nút ở các mức dưới chiếm đa số, cho nên việc phát sinh những nút ở trên nhiều lần cũng không có vấn đề gì.

Trong tìm kiếm với chiều sâu tăng dần, những nút ở dưới cùng (ở độ sâu d) được phát sinh một lần và những nút cao hơn thì được phát sinh hai lần và cứ thế đi lên đến gốc, được phát sinh d+1 lần. Vậy tổng số nút được phát sinh là:

$$N(\text{IDS}) = (d)b^1 + (d-1)b^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d.$$

So sánh với số lượng nút phát sinh bởi tìm kiếm theo chiều rộng là:

$$N(\text{BFS}) = b^1 + b^2 + \ldots + b^d + (b^{d+1} - b).$$

Lưu ý rằng, tìm kiếm theo chiều rộng có phát sinh một số nút ở độ sâu d+1 trong khi tìm kiếm lặp sâu dần thì không. Kết quả là tìm kiếm lặp sâu dần thực sự nhanh hơn, mặc dù những trạng thái được phát sinh lặp lại nhiều lần. Ví dụ với $b = 10$ và $d = 5$ ta có:

$$N(\text{IDS}) = 50 + 400 + 3000 + 20000 + 100000 = 123450.$$

$$N(\text{BFS}) = 10 + 100 + 1000 + 10000 + 100000 + 999990 = 1111100.$$

Độ phức tạp về mặt thời gian của tìm kiếm lặp sâu dần vẫn là $O(b^d)$ và về mặt không gian là $O(b \times d)$. Tóm lại, tìm kiếm với độ sâu tăng dần được sử dụng nhiều khi không gian tìm kiếm lớn và độ sâu của lời giải không xác định.

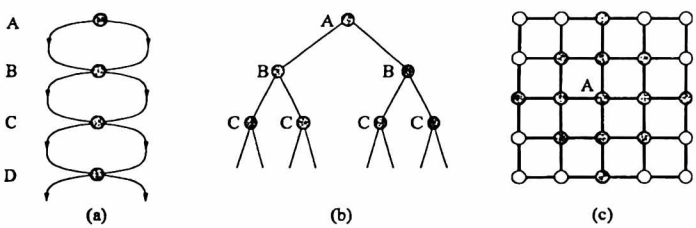
V. TRÁNH TRÙNG LẶP TRẠNG THÁI

Các phần trước đã đề cập qua một số thuật toán tìm kiếm nhưng lại bỏ qua trường hợp phức tạp trong quá trình tìm kiếm, đó là khả năng mở lại những trạng thái đã được xem xét và mở rộng trước đó. Với một số bài toán thì vấn đề này không xảy ra vì không gian trạng thái có dạng cây và chỉ có một đường đi duy nhất tới mỗi trạng thái. Chẳng hạn,

với một phát biểu của bài toán 8 quân hậu, là mỗi quân hậu mới được đặt tại cột trống đầu tiên tính từ bên trái thì mỗi trạng thái chỉ có thể đến được chỉ bằng một con đường duy nhất. Nếu phát biểu lại bài toán 8 quân hậu là một quân hậu có thể đặt vào bất kì cột nào thì mỗi trạng thái với n quân hậu có thể đến được bằng $n!$ con đường khác nhau.

Với một số bài toán, việc trùng lặp trạng thái là không tránh khỏi. Bao gồm những bài toán mà các hành động có thể đảo ngược được, như bài toán tìm đường đi hoặc bài toán 8-puzzle và dạng tổng quát của nó. Cây tìm kiếm của những bài toán này là vô hạn, nhưng nếu ta loại bỏ những trạng thái trùng lặp thì có thể đưa cây tìm kiếm về kích thước hữu hạn. Dễ dàng tìm ra những ví dụ mà việc loại bỏ các trạng thái trùng làm giảm đáng kể chi phí tìm kiếm.

Trong trường hợp xấu nhất, một không gian trạng thái kích thước $d+1$ (Hình 1.17(a)) có thể cho ra một cây tìm kiếm với 2^d nút lá (Hình 1.17(b)). Một ví dụ thực tế hơn là lưới ô vuông như trong Hình 1.17(c). Trên lưới mỗi trạng thái có bốn trạng thái con, như vậy cây tìm kiếm chứa những trạng thái trùng lặp có 4^d nút lá, nhưng chỉ có $2d^2$ trạng thái khác nhau trong phạm vi d bước từ một trạng thái bất kì. Với $d = 20$ sẽ có một nghìn tỉ nút, nhưng chỉ có khoảng 800 trạng thái khác nhau.



Hình 1.17 - *Những không gian trạng thái tạo ra những cây tìm kiếm rất lớn. (a) Một không gian trạng thái trong đó có hai khả năng biến đổi từ A sang B, hai khả năng từ B sang C và cứ thế tiếp tục. Không gian trạng thái chứa d + 1 trạng thái, với d là độ sâu tối đa. (b) Cây tìm kiếm tương ứng có 2^d nhánh tương ứng với 2^d đường đi trong không gian. (c) Một không gian lưới ô vuông. Những trạng thái tới được từ trạng thái (A) trong 2 bước được tô màu xám.*

ô vuông. Những trạng thái tới được từ trạng thái (A) trong 2 bước được tô màu xám.

Những trạng thái trùng lặp có thể làm cho một bài toán giải được trở thành không giải được nếu thuật toán không phát hiện ra. Cách phát hiện thông thường là so sánh nút sắp được mở rộng với những nút đã được mở rộng, xem giống nhau không, nếu giống nhau thì xác định được có hai đường đi đến cùng một trạng thái và ta có thể loại bỏ một trong hai trạng thái này.

Đối với tìm kiếm theo chiều sâu, chỉ những nút nằm trên đường đi từ nút gốc đến nút hiện tại là được lưu trong bộ nhớ. Việc so sánh nút hiện tại với những nút này cho phép thuật toán phát hiện chu trình trong đường đi và có thể loại bỏ ngay. Cách này sẽ bảo đảm không gian trạng thái hữu hạn, không tạo ra cây tìm kiếm vô hạn, tạo bởi các chu trình lặp lại, thật không may nó lại không tránh được trường hợp bùng nổ theo hàm mũ các đường đi không chu trình như trong bài toán mô tả trong Hình 1.17. Cách duy nhất để tránh điều này là lưu tất cả nút vào bộ nhớ. Về cơ bản có một sự đánh đổi giữa không gian và thời gian. Những thuật toán không ghi nhớ lại những gì đã xảy ra, thì sẽ chắc chắn bị lặp lại.

Nếu một thuật toán ghi nhớ tất cả trạng thái mà nó đã xem xét, thì có thể xem như thuật toán đó đã thực hiện tìm kiếm trực tiếp trên đồ thị không gian trạng thái. Có thể thêm một cấu trúc dữ liệu gọi là tập đóng, chứa các trạng thái đã mở rộng vào thuật toán tìm kiếm tổng quát trên cây, lúc này tập các trạng thái chưa được mở rộng gọi là tập mở. Nếu nút hiện tại đang xét trùng với một nút có trong tập đóng, nó sẽ bị loại bỏ thay vì đã được mở rộng. Thuật toán mới gọi là tìm kiếm tổng quát trên đồ thị (dựa trên bản chất của nó) như trong Hình 1.18.

Đối với những bài toán có nhiều trạng thái lặp lại thì tìm kiếm trên đồ thị nhanh hơn nhiều so với tìm kiếm trên cây. Độ phức tạp về thời gian và không gian của thuật toán tỉ lệ với kích thước của không gian trạng thái và nhỏ hơn rất nhiều so với $O(b^d)$.

Vấn đề tối ưu trong tìm kiếm trên đồ thị khá phức tạp. Như đã đề cập ở trên là khi gặp một trạng thái trùng lặp lại, tức là có hai con đường dẫn tới cùng một trạng thái, thuật toán mô tả ở trên luôn loại bỏ con

đường thứ hai mới tìm ra. Rõ ràng nếu con đường mới tìm ra này ngắn hơn con đường cũ thì thuật toán sẽ bỏ qua một lời giải tối ưu. May mắn là ta có thể chứng tỏ rằng điều này không thể xảy ra ngay cả đối với tìm kiếm chi phí đồng nhất hay tìm kiếm theo chiều rộng. Do hai thuật toán này cũng tối ưu khi sử dụng chiến lược tìm kiếm trên đồ thị. Mặt khác, tìm kiếm lặp sâu dần dựa trên tìm kiếm theo chiều sâu có thể dễ dàng đi vào con đường tối ưu cục bộ trước khi tìm thấy con đường tối ưu thực sự. Do đó, tìm kiếm lặp sâu dần trên đồ thị cần phải kiểm tra xem con đường mới tìm ra có tốt hơn con đường trước đó không, nếu có thì phải gán lại độ sâu và chi phí đường đi cho các nút con của trạng thái đó.

```
function Tìm_kiếm_Đồ_thị (bài_toán,ngăn_chứa) return lời giải hoặc thất bại
tập_đóng← khởi tạo rỗng
ngăn_chứa← Thêm(Tạo_nút(Trạng_thái_đầu[bài_toán]), ngăn_chứa)
loop do
    if Là_Rỗng(ngăn_chứa) then return thất bại.
    nút←Lấy_phần_tử_đầu(ngăn_chứa)
    if Kiểm_tra_Câu_hỏi_đích[bài_toán] trên Trạng_thái[nút] đúng
        then return Lời_giải(nút)
    if Trạng_thái[nút] không nằm trong tập_đóng then
    thêmTrạng_thái[nút] vào tập_đóng
    ngăn_chứa←Thêm_Tất_cả(Mở(nút,bài_toán), ngăn_chứa).
```

Hình 1.18 - *Thuật toán Tìm kiếm trên đồ thị.*

Lưu ý rằng, việc sử dụng tập đóng đồng nghĩa với việc tìm kiếm theo chiều sâu và tìm kiếm lặp sâu dần không còn tuyến tính nữa về mặt không gian sử dụng. Do thuật toán tìm kiếm tổng quát trên đồ thị lưu giữ tất cả các nút trong bộ nhớ chính, vì vậy mà một số tìm kiếm không thực hiện được bởi giới hạn về bộ nhớ.

VI. KẾT CHƯƠNG

Chương này giới thiệu các phương pháp giải quyết các bài toán bằng cách xây dựng một chuỗi các hành động để đạt được đến đích, quá trình này được gọi là *tìm kiếm*.

- Một thuật toán tìm kiếm trên cây tổng quát có thể được sử dụng để giải bất cứ bài toán nào; những biến thể của thuật toán chứa đựng những chiến lược tìm kiếm khác nhau.
- Các thuật toán tìm kiếm được đánh giá trên các tiêu chuẩn: tính đầy đủ, tính tối ưu, độ phức tạp về mặt thời gian và độ phức tạp về mặt không gian. Độ phức tạp phụ thuộc vào *b*, hệ số phân nhánh trong không gian trạng thái và *d* độ sâu của lời giải nông nhất.
- Tìm kiếm theo chiều rộng lựa chọn nút nông nhất chưa được mở trên cây tìm kiếm để mở rộng nó. Thuật toán là đầy đủ, tối ưu với chi phí bước đơn vị, và có độ phức tạp thời gian và không gian là $O(b^{d+1})$. Độ phức tạp về mặt không gian làm cho thuật toán không thực tế trong hầu hết trường hợp. Tìm kiếm chi phí đồng nhất tương tự tìm kiếm theo chiều rộng nhưng nó mở rộng nút có thi phí đường đi thấp nhất, ký hiệu *g(n)*. Thuật toán đầy đủ và tối ưu nếu chi phí của các bước lớn hơn 0.
- Tìm kiếm theo chiều sâu chọn nút sâu nhất chưa mở trên cây tìm kiếm để mở rộng nó. Thuật toán không đầy đủ lẫn tối ưu và có đội phức tạp thời gian là $O(b^m)$ và độ phức tạp không gian là $O(bm)$, trong đó *m* là độ sâu tối đa của bất kỳ đường đi nào trong không gian trạng thái.
- Tìm kiếm độ sâu giới hạn sử dụng một giới hạn độ sâu cố định với thuật toán tìm kiếm theo chiều sâu.
- Tìm kiếm lặp sâu dần còn gọi là tìm kiếm có độ sâu giới hạn, với giới hạn được tăng dần cho đến khi tìm thấy đích. Thuật toán đầy đủ, tối ưu cho các chi phí bước đơn vị và có độ phức tạp thời gian $O(b^d)$ và độ phức tạp không gian $O(bd)$.

VII.THUẬT NGỮ TIẾNG ANH

Sau đây là những thuật ngữ tiếng Anh liên quan đến những khái niệm đã dùng ở chương này:

action: hành động

backward pointer: con trỏ quay lui

branching factor: hệ số phân nhánh

breath first search (BFS): tìm kiếm theo chiều rộng

completeness: tính đầy đủ

depth first search (DFS): tìm kiếm theo chiều sâu

depth limited search: tìm kiếm với độ sâu giới hạn

fringe: nhánh non

goal state: trạng thái đích

initial state: trạng thái ban đầu

iterative deepening search: tìm kiếm lặp sâu dần

leaf node: nút lá

optimality: tính tối ưu

path cost: chi phí đường đi

priority queue: hàng đợi ưu tiên

problem: bài toán hay vấn đề

queue: hàng đợi

real-world problem: bài toán thực tế

search node: nút tìm kiếm

search problem: bài toán tìm kiếm

search tree: cây tìm kiếm

space complexity: độ phức tạp về không gian

stack: ngăn chứa

state: trạng thái

state space: không gian trạng thái

successor: trạng thái con

successor function: hàm trạng thái con

time complexity: độ phức tạp về thời gian

toy problem: bài toán trò chơi

uniform cost search (UCS): tìm kiếm chi phí đồng nhất

uniformed search, blind search: tìm kiếm không có thông tin, tìm kiếm mù

Chương 2

CÁC CHIẾN LƯỢC TÌM KIẾM CÓ THÔNG TIN

Chương 1 trình bày về các chiến lược tìm kiếm mù, có khả năng tìm ra lời giải cho bài toán bằng cách lần lượt phát sinh các trạng thái mới và kiểm tra xem có phải là trạng thái đích không. Tuy nhiên, những chiến lược này cực kì tốn kém trong hầu hết các trường hợp. Chương này trình bày về các chiến lược tìm kiếm có thông tin (dựa trên tri thức có được từ bài toán) có khả năng tìm ra lời giải hiệu quả hơn. Phần I mô tả các thuật toán tìm kiếm có thông tin và phần II đề cập đến cách để có được những tri thức cần thiết về bài toán. Cuối cùng, phần III bàn về các chiến lược tìm kiếm cục bộ trong không gian trạng thái, thay vì tìm kiếm trên từng con đường xuất phát từ trạng thái ban đầu, các chiến lược này sẽ thực hiện cùng lúc trên nhiều trạng thái hiện hành. Nhóm các thuật toán tìm kiếm cục bộ còn bao gồm những thuật toán lấy ý tưởng từ vật lý học (tìm kiếm mô phỏng luyện kim) và quá trình tiến hóa trong tự nhiên (thuật giải di truyền)

I. TÌM KIẾM CÓ THÔNG TIN

Các chiến lược tìm kiếm có thông tin dựa trên tri thức có được từ bài toán bên cạnh định nghĩa của bài toán để tìm ra lời giải hiệu quả hơn nhiều so với các chiến lược tìm kiếm mù. Cách tiếp cận tổng quát ta sẽ xem xét gọi là **tìm kiếm tối ưu**, nó dựa trên một **hàm đánh giá** để chọn ra nút tốt nhất để mở rộng. Thông thường hàm đánh giá này sẽ tính toán khoảng cách tới đích (còn bao xa nữa sẽ tới đích) như vậy sẽ chọn nút có giá trị khoảng cách nhỏ nhất.

Chữ "tối ưu" không có nghĩa là *lúc nào* cũng lấy được nút tốt nhất mà có nghĩa là lấy được nút *được xem là* tốt nhất theo tiêu chí của hàm

đánh giá. Nếu hàm đánh giá này phản ánh chính xác thì khi này nút được chọn sẽ thực sự tốt nhất, nếu không thì nó sẽ làm lệch hướng tìm kiếm. Có một họ rất nhiều thuật toán dạng này với những hàm đánh giá khác nhau. Điểm mấu chốt của những thuật toán này là ở **hàm *heuristic*** (kí hiệu $h(n)$) với $h(n)$ = xấp xỉ chi phí của đường đi (chi phí thấp nhất) từ nút thứ n đến nút đích.

Ví dụ: Có thể xấp xỉ bằng khoảng cách đường chim bay từ thành phố A đến thành phố B.

Các hàm *heuristic* là dạng thông dụng nhất để đưa tri thức về bài toán vào trong thuật toán tìm kiếm. Hàm *heuristic* được tìm hiểu sâu hơn trong phần II. Trong phần này ta xem chúng là các hàm đặc trưng cho bài toán với một ràng buộc là $h(n) = 0$ với n là nút đích. Phần còn lại sẽ đề cập đến hai cách để sử dụng thông tin từ hàm *heuristic* này.

I.1 Hàm *heuristic* – Tìm kiếm tối ưu kiểu tham lam (Greedy Best-First search).

Tìm kiếm tối ưu kiểu tham lam mở rộng nút gần đích nhất với hi vọng cách làm này sẽ dẫn đến lời giải một cách nhanh nhất. Do đó thuật toán này đánh giá chi phí của các nút chỉ dựa trên hàm *heuristic*:

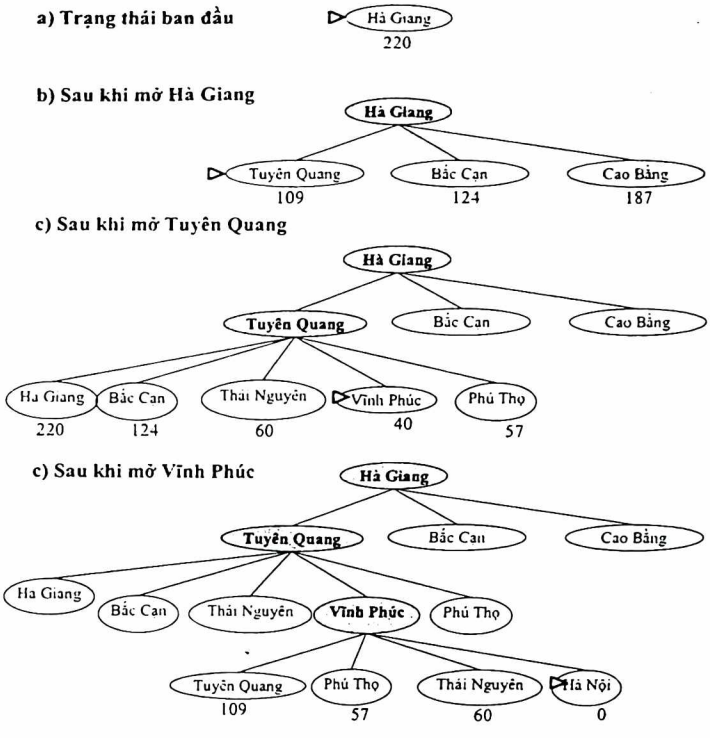
$$f(n) = h(n).$$

Xét bài toán tìm đường đi du lịch giữa các thành phố trong chương I, với hàm *heuristic* là **khoảng cách đường chim bay** đặt là h_{CB}. Giả sử đích là thành phố Hà Nội, ta cần xác định khoảng cách đường chim bay từ các thành phố khác tới Hà Nội (xem trong Bảng 2.1) Ví dụ h_{CB} (Ở(Hà Giang)) = 220. Lưu ý rằng giá trị h_{CB} không thể tính toán được từ mô tả của bài toán, cần phải có kinh nghiệm mới biết được h_{CB} có liên quan đến độ dài đường đi thực sự.

Bảng 2-1. *Giá trị h_{CB} – khoảng cách đường chim bay đến Hà Nội.*

Hà Giang	220	**Cao Bằng**	187
Bắc Cạn	124	**Tuyên Quang**	109
Thái Nguyên	60	**Lạng Sơn**	130
Phú Thọ	57	**Vĩnh Phúc**	40
Bắc Giang	46	**Hà Nội**	0
Bắc Ninh	27	**Hưng Yên**	48

Hải Dương	51	**Quảng Ninh**	141
Hà Nam	55	**Thái Bình**	83



Hình 2.1 - *Các bước trong tìm kiếm tham lam sử dụng heuristic h_{CB}. Các nút được đánh kèm giá trị h này.*

Hình 2.1 trình bày các bước của thuật toán tìm kiếm kiểu tham lam sử dụng giá trị h_{CB} để tìm ra đường đi từ Hà Giang đến thành phố Hà Nội. Nút được mở rộng đầu tiên từ Hà Giang là Tuyên Quang bởi vì

nó gần Hà Nội hơn Cao Bằng và Bắc Cạn. Nút được mở rộng tiếp theo là Vĩnh Phúc cũng vì nó gần đích hơn. Cuối cùng Vĩnh Phúc đi tới được đích là Hà Nội. Với ví dụ này, thuật toán tìm kiếm kiểu tham lam sử dụng h_{CB} tìm ra được lời giải mà không cần mở rộng thêm nút nào khác không có trong lời giải. Do đó, chi phí tìm kiếm của thuật toán là nhỏ nhất, nhưng nó lại không tối ưu. Đường đi qua Tuyên Quang và Vĩnh Phúc để đến Hà Nội dài hơn đường đi qua Phú Thọ 18 km. Điều này cũng cho thấy tại sao thuật toán này có tên là "*tham lam*", ở mỗi bước thuật toán chọn nút nào gần đích nhất mà không quan tâm đến điều gì khác.

Tìm kiếm kiểu tham lam giống tìm kiếm theo chiều sâu ở cách tìm kiếm chỉ đi theo một con đường để lần tới đích, còn nếu gặp ngõ cụt thì sẽ quay lại đi sang hướng khác. Thuật toán này cũng có khuyết điểm như tìm kiếm theo chiều sâu đó là: không tối ưu và không đầy đủ (vì nó có thể đi vào con đường vô tận và không bao giờ quay trở lại được). Độ phức tạp về thời gian và không gian là $O(b^m)$ với m là độ sâu tối đa của không gian tìm kiếm. Tuy nhiên với hàm *heuristic* tốt, độ phức tạp có thể giảm đi đáng kể. Mức giảm tùy thuộc vào từng bài toán cụ thể và chất lượng của hàm *heuristic*.

I.2 Tìm kiếm A*

Một thuật toán tìm kiếm tối ưu được biết đến nhiều nhất đó chính là tìm kiếm A*. Thuật toán này đánh giá một nút dựa trên chi phí đi từ nút gốc đến nút đó – $g(n)$, cộng với chi phí từ nút đó đến đích – $h(n)$.

$$f(n) = g(n) + h(n).$$

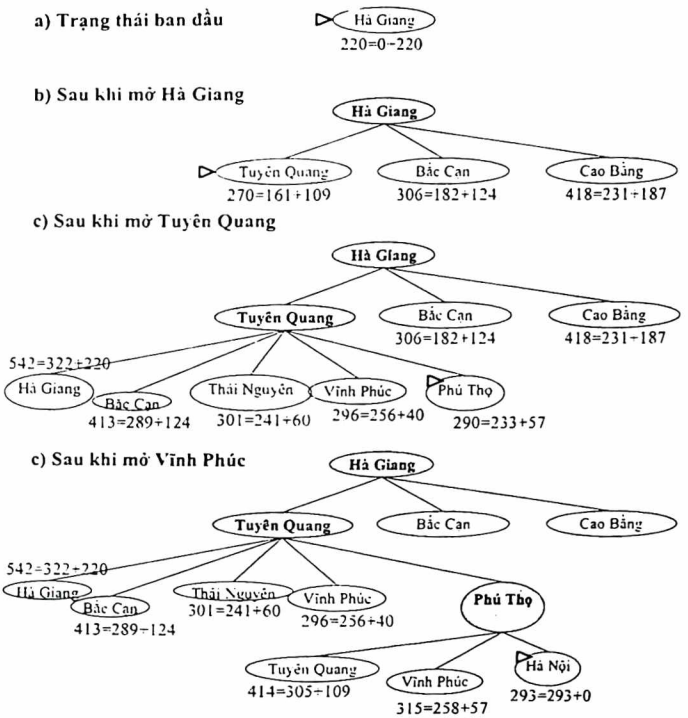
Do giá trị $g(n)$ cho biết chi phí đường đi từ nút gốc đến nút n và $h(n)$ là ước lượng chi phí đường đi ngắn nhất từ nút n đến đích nên ta có:

$f(n)$ = ước lượng chi phí của lời giải "tốt nhất" qua n.

Do đó, nếu ta định tìm lời giải "tốt nhất" thì cách dễ thấy nhất là lấy nút có giá trị: $g(n) + h(n)$ nhỏ nhất. Hơn nữa nếu hàm *heuristic* $h(n)$ thỏa một số điều kiện nhất định thì tìm kiếm A* sẽ đầy đủ và tối ưu.

Để bảo đảm A* tìm được lời giải tối ưu thì hàm $h(n)$ phải là hàm *heuristic* chấp nhận được. Hàm *heuristic* chấp nhận được khi nó không bao giờ *ước lượng quá* chi phí để đến đích thực sự. Điều này sẽ

đảm bảo sự tối ưu vì luôn xem chi phí để giải bài toán nhỏ hơn chi phí thực sự. Do $g(n)$ là chi phí thực sự để đến nút n, từ đó suy ra $f(n)$ không bao giờ ước lượng quá chi phí thực sự của một lời giải qua n.



Hình 2.2 - *Các bước trong thuật toán A*. Các nút được đánh kèm giá trị $f = g + h$. Giá trị h là khoảng cách đường chim bay tới Hà Nội từ Bảng 2-1*

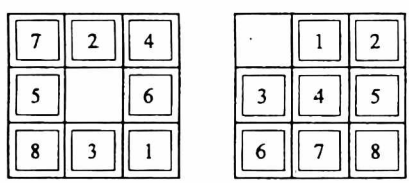
Một ví dụ cụ thể về một hàm *heuristic* chấp nhận được, chính là khoảng cách đường chim bay sử dụng trong bài toán tìm đường đi du lịch giữa các thành phố. Khoảng cách này chấp nhận được, vì đường đi ngắn nhất giữa hai điểm chính là đường thẳng nối hai điểm này, do đó khoảng cách này không thể nào lớn hơn đường đi thực sự được.

Hình 2.2 cho thấy quá trình xây dựng cây tìm kiếm của thuật toán A* với đích là Hà Nội. Qua ví dụ, có một chứng minh tổng quát rằng thuật toán A* sẽ tối ưu nếu $h(n)$ chấp nhận được. Giả sử G2 là nút đích chưa tối ưu và chi phí của lời giải tối ưu là C*. Do G2 chưa tối ưu và h(G2) = 0 (G2 là nút đích) nên:

$f(G2) = g(G2) + h(G2) = g(G2) > C^*$. Gọi n là một nút lá nằm trên đường đi tối ưu, chẳng hạn như Phú Thọ. Nếu $h(n)$ không ước lượng quá chi phí đường đi thực sự thì:

$f(n) = g(n) + h(n) \leq C^*$.

Từ đây ta suy ra được: $f(n) \leq C^* \leq g$(G2), như vậy G2 sẽ không được mở rộng và A* luôn tìm ra lời giải tối ưu.



Hình 2.3 - *Một thể hiện của bài toán 8-puzzle. Lời giải gồm 26 bước.*

II. HÀM *HEURISTIC*.

Trong phần này, ta sẽ xem xét *heuristic* trong bài toán 8-puzzle, để thấy được bản chất chung của *heuristic*.

Bài toán 8-puzzle là một trong những bài toán đầu tiên sử dụng tìm kiếm *heuristic*. Như đã đề cập trong phần I chương 1, mục tiêu của trò

chơi này là đẩy các quân cờ theo chiều ngang và chiều dọc tới khi có được cách sắp xếp mong muốn (Hình 2.3).

Chi phí trung bình của lời giải cho 8-puzzle phát sinh ngẫu nhiên là khoảng 22 bước. Hệ số phân nhánh trung bình là 3 (khi ô trống nằm ở giữa thì có bốn quân có thể đi được, khi ô trống ở góc thì có hai quân và khi nó ở dọc cạnh thì có ba quân). Điều này đồng nghĩa với việc tìm kiếm toàn diện với độ sâu 22, trên không gian có khoảng: $3^{22} \approx 3.1 \times 10^{10}$ trạng thái, nếu loại đi những trạng thái trùng lắp thì giảm xuống còn khoảng 170.000, vì chỉ có 9!/2 = 181.440 trạng thái khác nhau có thể tới được từ một trạng thái ban đầu. Với bài toán 15-puzzle, số trạng thái sẽ lên tới 10^{13}, như vậy công việc cần làm tiếp theo là tìm một hàm *heuristic* tốt. Nếu muốn tìm lời giải có số bước ngắn nhất bằng thuật toán A*, cần một hàm *heuristic* không bao giờ ước lượng qua số bước đến đích. Sau đây là hai hàm *heuristic* thường dùng:

- h_1 = *số lượng quân đặt sai vị trí.* Với Hình 2.3, tất cả 8 quân cờ đều không ở đúng vị trí mong muốn, do đó trạng thái bắt đầu sẽ có $h_1 = 8$. h_1 là một *heuristic* chấp nhận được vì bất kì quân này bị sai vị trí sẽ phải di chuyển ít nhất một lần.
- h_2 = *tổng khoảng cách từ quân cờ đến vị trí đích của nó.* Bởi vì, quân cờ không thể di chuyển theo đường chéo, khoảng cách được tính bằng tổng khoảng cách theo chiều ngang và chiều dọc. Khoảng cách này có khi được gọi là khoảng cách Manhattan. h_2 cũng chấp nhận được bất kì bước đi có thể nào đều di chuyển một quân tiến một bước gần tới đích hơn. Khoảng cách Manhattan của trạng thái bắt đầu tính theo thứ tự từ quân số 1 đến số 8:

$$h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18.$$

Cả hai hàm *heuristic* này đều không ước lượng quá chi phí thực sự, trong trường hợp của Hình 2.3 là 26.

..1 Ảnh hưởng của hàm *heuristic* lên hiệu quả

Một cách để đánh giá chất lượng của hàm *heuristic* là dựa trên hệ số phân nhánh trung bình hiệu quả b^*. Nếu tổng số nút được phát sinh

bởi thuật toán A* với một bài toán cụ thể là N và độ sâu của lời giải là d thì b^* là hệ số phân nhánh trung bình mà một cây đồng nhất chiều cao d phải có để chứa được $N + 1$ nút. Do đó :

$$N + 1 = 1 + b^* + \left(b^*\right)^2 + \ldots + \left(b^*\right)^d$$

Ví dụ, nếu A* tìm ra lời giải ở độ sâu 5 sử dụng 52 nút thì hệ số phân nhánh trung bình hiệu quả là 1.92. Giá trị này có thể thay đổi theo từng thể hiện của bài toán, nhưng thông thường sẽ không là hằng số với bài toán đủ khó. Do đó thử nghiệm độ đo b^* trên một tập nhỏ của bài toán có thể giúp ích cho đánh giá chung độ hữu ích của *heuristic*. Một *heuristic* được xây dựng tốt sẽ cho giá trị b^* gần bằng 1, cho phép giải những bài toán khá lớn.

Để kiểm tra hàm *heuristic* h_1 và h_2, phát sinh ngẫu nhiên 1.200 bài với lời giải có chiều dài từ 2 đến 24 (100 bài cho mỗi giá trị chẵn) và giải những bài này bằng tìm kiếm lặp sâu dần và tìm kiếm A* sử dụng h_1 và h_2. Bảng 2-2 cho thấy số lượng nút trung bình được phát sinh bởi mỗi thuật toán và hệ số phân nhánh trung bình hiệu quả. Kết quả cho thấy h_2 tốt hơn h_1 và tốt hơn nhiều so với tìm kiếm lặp sâu dần. Trong lời giải chiều dài 14, A* sử dụng h_2 hiệu quả gấp 30.000 lần so với tìm kiếm lặp sâu dần.

Người ta có thể hỏi có phải h_2 luôn tốt hơn h_1 không ? Câu trả lời là có. Điều này rất dễ thấy ngay trong định nghĩa của hai hàm *heuristic*, với nút n bất kì $h_2(n) \geq h_1(n)$. Ta có thể nói rằng h_2 **tốt hơn** h_1. Sự tốt hơn thể hiện qua hiệu suất: A* sử dụng h_2 sẽ không bao giờ mở rộng nhiều nút hơn A* sử dụng h_1 (trừ trường hợp với các nút có $f(n) = C^*$). Điều này được chứng minh như sau: tất cả các nút có chi phí $f(n) < C^*$ sẽ chắc chắn được mở rộng và tương tự, mọi nút có $h(n) < C^* - g(n)$ sẽ chắc chắn được mở rộng. Nhưng bởi vì ít nhất h_2 cũng gần bằng h_1 với mọi nút, với bất cứ nút nào chắc chắn được mở rộng bởi tìm kiếm A* với h_2 cũng sẽ được mở rộng với h_1 và h_1 cũng có thể làm cho một số nút khác được mở rộng luôn. Do đó, sẽ luôn tốt hơn khi sử dụng một hàm *heuristic* có giá trị cao hơn, nếu nó không ước lượng quá và thời gian tính toán *heuristic* không quá lớn.

Bảng 2-2. *So sánh chi phí tìm kiếm và hệ số phân nhánh trung bình hiệu quả cho thuật toán tìm kiếm lặp sâu dần và A* với h_1 và h_2. Dữ liệu tính trung bình trên 100 thể hiện của bài toán 8-puzzle ứng với mỗi độ sâu lời giải.*

	Chi phí tìm kiếm			Số nhánh trung bình hiệu quả		
d	ID	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	1	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	36440	227	73	2.78	1.42	1.24
14	-	539	113	-	1.44	1.23
16	-	1301	211	-	1.45	1.25
18	-	3056	363	-	1.46	1.26
20	-	7276	676	-	1.47	1.27
22	-	18094	1219	-	1.48	1.28
24	-	39135	1641	-	1.48	1.26

II.2 Phát minh ra các hàm *heuristic* chấp nhận được

Ta đã thấy rằng cả h_1 (số lượng quân sai vị trí) và h_2 (khoảng cách Manhattan) là những *heuristic* khá tốt cho 8-puzzle và h_2 tốt hơn. Người ta nghĩ ra h_2 thế nào? Có thể nào để máy tính tự phát minh ra một hàm *heuristic* như vậy không?

h_1 và h_2 ước lượng chiều dài còn lại của đường đi cho bài toán 8-puzzle, nhưng chúng cũng cũng hoàn toàn chính xác là chiều dài đường đi cho phiên bản đơn giản của bài toán này. Nếu luật được sửa lại là cho phép di chuyển quân đi bất kì nơi nào mà không cần phải sang ô trống kế cận, thì khi đó h_1 sẽ cho biết chính xác số bước đi của lời giải ngắn nhất. Tương tự như vậy nếu cho phép quân cờ được di chuyển theo hướng bất kì, thậm chí đi vào một ô đã có quân, thì h_2 sẽ cho biết chính xác số bước đi của lời giải ngắn nhất. Một bài toán có ít giới hạn hơn trên những hành động có thể được gọi là **bài toán giản lược**. Chi phí của lời giải tối ưu trong bài toán giản lược là một hàm *heuristic* chấp nhận được trong bài toán ban đầu. Hàm *heuristic* chấp nhận được vì lời giải tối ưu trong bài toán ban đầu cũng chính là lời giải trong bài toán giản lược (có được do định nghĩa) và do đó ít nhất phải bằng với lời giải tối ưu trong bài toán giản lược.

Nếu định nghĩa của của một bài toán được viết lại một cách hình thức thì có thể tự động xây dựng ra các bài toán giản lược. Ví dụ, nếu các hành động trong 8-puzzle được mô tả như sau:

Một quân cờ có thể di chuyển từ ô A sang ô B nếu A là ô kế cận của B theo chiều ngang hoặc chiều dọc và B là một ô trống.

Ta có thể phát sinh ra ba bài toán giản lược bằng cách bỏ đi một hoặc cả ba điều kiện trên:

(a) Một quân cờ có thể di chuyển từ ô A sang ô B nếu A là ô kế cận của B.

(b) Một quân cờ có thể di chuyển từ ô A sang ô B nếu B là một ô trống.

(c) Một quân cờ có thể di chuyển từ ô A sang ô B.

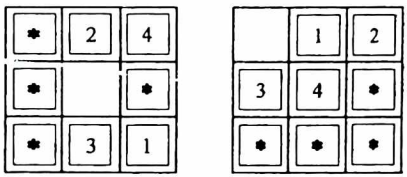
Từ (a) ta có được h_2 (khoảng cách Manhattan). Lý do có được là vì h_2 sẽ đúng khi ta di chuyển mỗi quân theo hàng ngang hay dọc để về tới vị trí của nó. Hàm *heuristic* có được từ (b) là *heuristic Gaschnig*. Từ (c) ta có được h_1 (số quân đặt sai), bởi vì sẽ phù hợp khi chỉ cần một bước là có thể di chuyển tới vị trí mong muốn. Lưu ý một điều quan trọng là bài toán giản lược được phát sinh bởi kĩ thuật này có thể giải một cách dễ dàng mà không cần tìm kiếm, vì những qui luật đã giản lược cho phép bài toán có thể được tách thành tám bài toán con độc lập. Nếu bài toán giản lược khó giải được, thì giá trị của *heuristic* tương ứng sẽ tốn kém chi phí để tính toán.

Heuristic chấp nhận được cũng có thể có được từ chi phí lời giải của một **bài toán con** của một bài toán cho trước. Ví dụ, Hình 2.4 cho thấy một bài toán con của 8-puzzle thể hiện trong Hình 2.3. Bài toán con thực hiện việc đưa các quân 1, 2, 3, 4 về vị trí đúng của chúng.

Rõ ràng chi phí cho lời giải tối ưu của bài toán con này là chặn dưới của chi phí của bài toán đầy đủ. Hàm *heuristic* này chính xác hơn khoảng cách Manhattan trong một số trường hợp.

Việc lựa chọn 1-2-3-4 là tùy ý, ta có thể xây dựng cơ sở dữ liệu cho 5-6-7-8 và cho 2-4-6-8, v.v... Mỗi cơ sở dữ liệu cho một hàm *heuristic* chấp nhận được và những *heuristic* này có thể được kết hợp lại bằng cách lấy giá trị lớn nhất. *Heuristic* tổng hợp dạng này chính xác hơn

nhiều so với khoảng cách Manhattan, số lượng nút được phát sinh khi giải bài toán 15-puzzle có thể giảm xuống 1.000 lần.



Hình 2.4 - *Một bài toán con của 8-puzzle thể hiện trong Hình 2.3. Công việc cần làm là đưa quân 1, 2, 3, 4 về đúng vị trí của chúng, mà không cần quan tâm đến những quân khác.*

III. TÌM KIẾM CỤC BỘ VÀ BÀI TOÁN TỐI ƯU

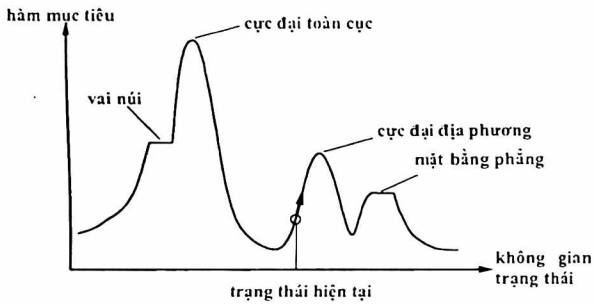
Những thuật toán tìm kiếm mà ta đã xem xét cho tới thời điểm này được thiết kế để duyệt trong không gian tìm kiếm một cách có hệ thống. Tính hệ thống đạt được bằng cách giữ trong bộ nhớ một hoặc nhiều đường đi và bằng cách ghi nhận cái nào đã được xem xét trên đường đi và cái nào chưa. Khi đích được tìm thấy, đường đi tới đích này cũng chính là lời giải cho bài toán.

Tuy nhiên, trong nhiều bài toán đường đi tới đích lại không có liên quan gì. Ví dụ trong bài toán 8 hậu, điều quan trọng là vị trí các quân hậu cuối cùng chứ không phải là thứ tự các quân này được thêm vào. Lớp bài toán này bao gồm nhiều ứng dụng quan trọng như thiết kế mạch, thiết kế qui trình sản xuất, lập lịch công việc, lập trình tự động, tối ưu mạng truyền thông, phân luồng xe cộ và quản lý danh mục đầu tư.

Nếu đường đi tới đích không quan trọng, ta có thể xem xét một lớp thuật toán khác không quan tâm đến đường đi nào cả. Thuật toán **tìm kiếm cục bộ** sử dụng một **trạng thái hiện tại** đơn lẻ (thay vì nhiều đường đi khác nhau) và chỉ di chuyển sang trạng thái lân cận. Thông thường các đường đi trong quá trình tìm kiếm không được giữ lại.

Mặc dù thuật toán tìm kiếm cục bộ không hệ thống nhưng lại có hai ưu điểm nổi bật: (1) sử dụng rất ít bộ nhớ thông thường là lượng cố định và (2) thường tìm thấy lời giải hợp lí trong không gian trạng thái lớn hoặc vô hạn (liên tục) mà thuật toán tìm kiếm có hệ thống không phù hợp.

Bên cạnh việc tìm kiếm đích, thuật toán tìm kiếm cục bộ cũng hữu ích cho việc giải các **bài toán tối ưu hóa** thuần túy với mục tiêu là tìm trạng thái "tốt nhất" căn cứ vào **hàm mục tiêu**. Ví dụ, trong tự nhiên có một hàm mục tiêu - độ thích nghi - theo như thuyết tiến hóa Darwin có thể xem như một bài toán tối ưu hóa, nhưng lại không hề có "kiểm tra là đích" và "chi phí đường đi" cho bài toán này.



Hình 2.5 - *Địa hình một chiều của không gian trạng thái trong đó việc đánh giá tương ứng với hàm mục tiêu. Mục tiêu là tìm cực đại toàn cục. Tìm kiếm leo đồi thay đổi trạng thái hiện tại để cải thiện nó, như hình mũi tên thể hiện. Những đặc trưng khác của địa hình được mô tả ở trên.*

Để hiểu được tìm kiếm cục bộ, ta sẽ xem xét **địa hình của không gian trạng thái** (như trong Hình 2.5) Một địa hình có cả "vị trí" (định nghĩa bởi trạng thái) và "cao độ" (định nghĩa bởi giá trị của hàm *heuristic* hoặc hàm mục tiêu). Nếu cao độ tương ứng với chi phí thì mục tiêu sẽ là tìm đồi thấp nhất - **cực tiểu toàn cục**. Nếu cao độ tương ứng với hàm mục tiêu là tìm đỉnh cao nhất - **cực đại toàn cục**. (có thể chuyển đổi giữa hai cái này bằng cách thêm dấu trừ) Thuật toán tìm kiếm cục bộ duyệt trong địa hình. Một thuật toán tìm kiếm cục bộ **đầy**

đủ luôn luôn tìm được đích nếu nó tồn tại, thuật toán là **tối ưu** khi nó luôn tìm ra cực đại/cực tiểu toàn cục.

III.1 Tìm kiếm leo đồi

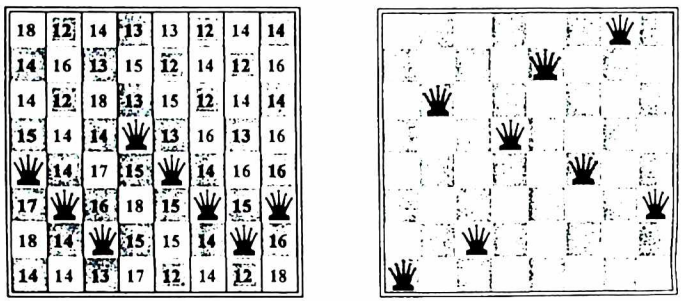
Thuật toán tìm kiếm leo đồi như sau:

```
function Leo_đồi (bài_toán) return trạng thái là cực đại cục bộ
    input:bài_toán
    biến cục bộ:   nút hiện_tại
                   một nút láng_giềng
    hiện_tại←Tạo_nút(Trạng_thái_đầu[bài_toán])
    loop do
        láng_giềng← nút con của hiện_tại có giá trị lớn nhất
        if Giá_trị[láng_giềng] ≤ Giá_trị [hiện_tại] then
            return Trạng_thái[hiện_tại]
        hiện_tại←nút lận cận
```

Hình 2.6 - *Thuật toán tìm kiếm leo đồi.*

Thuật toán chỉ đơn giản là một vòng lặp trong đó liên tục di chuyển theo hướng tăng dần giá trị tức là leo đồi. Thuật toán dừng khi tới được "đỉnh" nơi mà không có lân cận nào của nó có giá trị cao hơn. Thuật toán không duy trì một cây tìm kiếm, nên cấu trúc dữ liệu cho nút hiện tại chỉ cần lưu trạng thái và giá trị hàm mục tiêu của nó. Leo đồi không có xem xét xa hơn bên ngoài lân cận của trạng thái hiện tại.

Để minh họa cho thuật toán leo đồi, ta sử dụng bài toán 8 quân hậu đã giới thiệu ở chương 1. Thuật toán tìm kiếm cục bộ thông thường sử dụng một công thức cho trạng thái hoàn tất, trong đó mỗi trạng thái có 8 quân hậu trên bàn cờ, mỗi cột có một quân. Hàm trạng thái con trả về tất cả trạng thái có thể phát sinh được bằng cách di chuyển một quân hậu sang ô khác trong cùng một cột (mỗi trạng thái có $8 \times 7 = 56$ trạng thái con). Hàm *heuristic h* là số lượng các cặp quân hậu có thể tấn công lẫn nhau, trực tiếp hoặc gián tiếp. Cực tiểu toàn cục của hàm này là 0, chỉ xảy ra đối với lời giải đúng. Hình 2.7(a) cho thấy một trạng thái với $h = 17$. Hình này cũng cho thấy giá trị của các trạng thái con của nó, với trạng thái con tốt nhất có $h = 12$. Thuật toán leo đồi thông thường chọn ngẫu nhiên trong tập các trạng thái con một trạng thái tốt nhất, nếu có nhiều hơn một trạng thái.



(a) (b)

Hình 2.7 - (a) *Một trạng thái của 8 hậu với chi phí* heuristic *trước lượng h = 17, thể hiện các giá trị của h cho mỗi trạng thái con có được bằng cách di chuyển một quân hậu trong cùng một cột. Những bước tốt nhất được vẽ khung viền.* (b) *Một cực tiểu cục bộ trong không gian trạng thái 8 hậu, trạng thái này có h = 1, trong khi các trạng thái con của nó có giá trị cao hơn.*

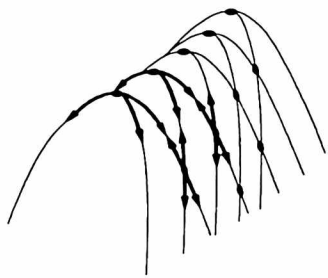
Thuật toán leo đồi đôi khi còn được gọi là **tìm kiếm cục bộ tham lam** vì nó lấy một trạng thái lân cận tốt mà không cần suy nghĩ thêm sau đó sẽ đi đâu tiếp. Thuật toán leo đồi thường tiến tới lời giải rất nhanh vì nó rất dễ cải thiện một trạng thái xấu. Ví dụ, từ trạng thái trong Hình 2.7(a), chỉ cần 5 bước là có thể tới được trạng thái ở Hình 2.7(b) với $h = 1$ và gần như là một lời giải. Thật không may thuật toán leo đồi thường bị kẹt do những lý do sau:

Cực đại cục bộ là một đỉnh mà nó cao hơn những trạng thái lân cận của nó, nhưng lại thấp hơn cực đại toàn cục. Thuật toán leo đồi tới được vùng lân cận của cực đại cục bộ sẽ bị dẫn dần tới đỉnh, nhưng sau đó sẽ bị kẹt ở đó và không biết đi đâu nữa. Hình 2.8 thể hiện vấn đề này. Cụ thể hơn, trạng thái trong Hình 2.7(b) thực chất là một cực đại cục bộ (cực tiểu cục bộ đối với chi phí h) khi di chuyển bất cứ quân hậu nào đều làm cho tình huống xấu hơn.

Chóp được thể hiện trong Hình 2.8. Chóp là kết quả của một loạt các cực đại cục bộ mà thuật toán tham lam khó biết được.

Mặt bằng phẳng là một vùng trong địa hình của không gian trạng thái nơi mà hàm đánh giá phẳng (giá trị bằng nhau). Nó có thể là cực đại cục bộ nơi không có đường đi lớn nữa hoặc là một bờ núi nơi có thể đi lên tiếp được (xem Hình 2.5) Quá trình tìm kiếm leo đồi có thể sẽ không tìm được lối thoát khỏi mặt bằng phẳng này.

Trong mỗi trường hợp, thuật toán đều đi tới một điểm mà nó không thể tiếp tục nữa. Bắt đầu từ một trạng thái 8 hậu phát sinh ngẫu nhiên, thuật toán leo đồi bị kẹt tới 86%, chỉ giải được 14% thể hiện của bài toán. Thuật toán chạy nhanh, tính trung bình chỉ cần 4 bước khi thành công và 3 bước khi thất bại, kết quả không tồi cho một không gian trạng thái có $8^8 \approx 17$ triệu trạng thái.



Hình 2.8 - *Minh họa tại sao đỉnh chóp lại gây khó khăn cho tìm kiếm leo đồi. Các trạng thái trên đỉnh chóp tăng dần từ trái sang phải, tạo ra một chuỗi các cực đại cục bộ không nối trực tiếp với nhau. Với mỗi cực đại cục bộ, tất cả những hành động có thể đều đi xuống đồi.*

Thuật toán dừng lại nếu nó gặp mặt bằng phẳng khi mà trạng thái con tốt nhất có cùng giá trị với trạng thái hiện tại. Để có thể tiếp tục việc tìm kiếm, ta có thể cho phép di chuyển ngang với hi vọng rằng mặt bằng phẳng thực sự là một bờ núi như trong Hình 2.5 tuy nhiên phải cân thận. Nếu ta luôn cho phép di chuyển ngang khi gặp trường hợp không có đồi để leo lên, vòng lặp vô tận sẽ xảy ra khi thuật toán gặp một mặt phẳng cực đại cục bộ không phải là bờ núi. Một giải pháp đơn giản là đặt ngưỡng giới hạn số lần di chuyển ngang liên tục. Ví dụ, ta có thể cho phép di chuyển ngang liên tục 100 lần trong bài toán

8 hậu. Việc này làm tăng tỉ lệ giải được bài toán của thuật toán leo đồi từ 14% lên 94%. Thành công này đi kèm với một cái giá phải trả là thuật toán cần trung bình 21 bước cho các thể hiện giải thành công và 64 cho những thể hiện thất bại.

Rất nhiều biến thể của thuật toán leo đồi đã được đưa ra. Thuật toán **tìm kiếm leo đồi ngẫu nhiên** chọn ngẫu nhiên các bước leo đồi, xác suất chọn có thể thay đổi theo độ dốc của bước leo đồi. Điều này sẽ làm thuật toán hội tụ chậm hơn so với thuật toán gốc, nhưng trong một số địa hình nó tìm ra lời giải tốt hơn. Thuật toán **tìm kiếm leo đồi có lựa chọn** cài đặt lại thuật toán leo đồi ngẫu nhiên bằng cách phát sinh trạng thái con ngẫu nhiên cho tới khi có một trạng thái con tốt hơn trạng thái hiện tại. Đây là một chiến lược tốt khi một trạng thái có nhiều (hàng ngàn chẳng hạn) trạng thái con.

Các thuật toán leo đồi mô tả tới lúc này vẫn không đầy đủ, thường thất bại trong việc tìm tới đích khi nó tồn tại bởi vì các thuật toán có thể bị kẹt ở cực đại cục bộ. Thuật toán **leo đồi làm lại ngẫu nhiên** tiến hành một loạt các quá trình tìm kiếm leo đồi từ trạng thái phát sinh ngẫu nhiên ban đầu, dừng khi đích được tìm thấy. Nó đầy đủ khi xác suất tiến gần 1, với lý do đơn giản là nó sẽ phát sinh ra được trạng thái ban đầu chính là trạng thái đích luôn. Nếu mỗi quá trình tìm kiếm leo đồi có xác suất thành công là p thì số lần cần làm lại sẽ là $1/p$. Với bài toán 8 hậu không cho phép di chuyển ngang, $p \approx 0.14$, nên ta cần khoảng 7 lần lặp để tìm ra đích (6 lần thất bại và 1 lần thành công). Số bước mong đợi chính là chi phí của một lần lặp thành công cộng với $(1-p)/p$ lần thất bại, vào khoảng 22 bước. Khi ta cho phép di chuyển ngang, cần trung bình $1/0.94 \approx 1.06$ lần lặp và $(1 \times 21) + (0.06/0.94) \times 64 \approx 25$ bước. Với bài 8 hậu thì thuật toán này thực sự hiệu quả. Thậm chí với ba triệu quân hậu, cách làm này có thể tìm ra lời giải trong thời gian dưới một phút.

Sự thành công của thuật toán leo đồi phụ thuộc rất nhiều vào hình dạng địa hình của không gian trạng thái: nếu có rất ít cực đại cục bộ và mặt bằng phẳng, leo đồi làm lại ngẫu nhiên sẽ tìm ra một lời giải tốt rất nhanh. Ngược lại, nhiều bài toán thực tế có vô số cực đại cục bộ. Ví dụ, các bài toán NP- khó thông thường có một số lượng cực đại cục

bộ theo hàm mũ. Dù vậy thì một cực đại cục bộ thường có thể được tìm thấy sau một số ít lần thực hiện lại.

III.2 Tìm kiếm nhóm cục bộ

Thuật toán **tìm kiếm nhóm cục bộ** theo dõi k trạng thái thay vì chỉ một. Nó bắt đầu với k trạng thái được phát sinh ngẫu nhiên. Ở mỗi bước, tất cả trạng thái con của k trạng thái này được phát sinh. Nếu bất kì trạng thái nào là đích thì thuật toán dừng. Ngược lại, nó chọn k trạng thái con tốt nhất từ danh sách trạng thái con mới này và lặp lại công việc này.

Tìm kiếm nhóm cục bộ với k trạng thái dường như không khác gì với việc chạy k tìm kiếm làm lại ngẫu nhiên song song thay vì chạy tuần tự. Trên thực tế, hai thuật toán khá khác nhau. Trong tìm kiếm khởi tạo ngẫu nhiên, mỗi quá trình tìm kiếm chạy một cách độc lập với nhau. Trong tìm kiếm nhóm cục bộ, *thông tin hữu ích được truyền qua lại giữa k luồng tìm kiếm song song*. Ví dụ nếu một trạng thái phát sinh ra các trạng thái con tốt và $k - 1$ trạng thái còn lại phát sinh ra những trạng thái con không tốt, thì hiệu ứng ở đây là trạng thái đầu tiên sẽ thu hút các trạng thái khác về phía mình. Thuật toán nhanh chóng bỏ đi những hướng tìm kiếm không tốt và chuyển sang nơi có tiến triển tốt nhất.

III.3 Tìm kiếm mô phỏng luyện kim

Thuật toán leo đồi không bao giờ thực hiện bước "xuống đồi" đến những trạng thái có giá trị thấp hơn (hoặc chi phí cao hơn) chắc chắn không đầy đủ, vì thế nó có thể bị kẹt ở cực đại cục bộ. Trái lại, cách đi hoàn toàn ngẫu nhiên (tức là chuyển tới trạng thái con được chọn ngẫu nhiên từ tập các trạng thái con) sẽ đầy đủ nhưng lại cực kì không hiệu quả. Do đó, sẽ là một điều hợp lí khi ta thử kết hợp leo đồi với di chuyển ngẫu nhiên theo cách nào đó như vậy sẽ có được cả sự hiệu quả và tính đầy đủ. **Tìm kiếm mô phỏng luyện kim** là một thuật toán như vậy. Trong công nghiệp luyện kim, **luyện kim** là quá trình được sử dụng để tôi luyện kim loại và thủy tinh bằng cách nung nóng chúng ở nhiệt độ cao và sau đó làm lạnh từ từ, từ đó cho phép vật liệu kết dính lại ở trạng thái kết tinh năng lượng thấp. Thuật toán mô phỏng luyện kim có thể được hình dung như khi ta thả một quả bóng xuống

một bề mặt mấp mô. Nếu ta để quả bóng tự lăn thì nó sẽ dừng ở cực tiểu cục bộ. Nếu ta lắc bề mặt thì ta có thể tung quả bóng ra khỏi cực tiểu cục bộ. Mẹo ở đây là lắc đủ mạnh để tung trái bóng ra khỏi cực tiểu cục bộ, nhưng không quá mạnh để đánh bật nó ra khỏi cực tiểu toàn cục. Giải pháp mô phỏng luyện kim là bắt đầu bằng việc lắc mạnh (ứng với nhiệt độ cao) và sau đó từ từ giảm cường độ lắc (giảm dần nhiệt độ).

```
function Mô_phỏng_Luyện_kim(bài_toán, lịch) return lời giải
    input: bài_toán,
           lịch, ánh xạ giữa thời gian và "nhiệt độ"
    biến cục bộ: nút hiện_tại,
           nút kế_tiếp,
           T, "nhiệt độ" điều khiển xác suất các bước xuống dốc
    hiện_tại ← Tạo_Nút(Trạng_thái_đầu[bài_toán])
    for t ← 1 to ∞ do
           T ← lịch[t]
           if T = 0 then return hiện_tại
           kế_tiếp ← nút con được chọn ngẫu nhiên của hiện_tại
           ΔE ← Giá_trị[kế_tiếp] – Giá_trị [hiện_tại]
           if ΔE > 0 then hiện_tại ← kế_tiếp
           else hiện_tại ← kế_tiếp chỉ với xác suất e^(ΔE/T)
```

Hình 2.9 - *Thuật toán Mô phỏng Luyện kim.*

Vòng lặp trong cùng nhất của thuật toán mô phỏng luyện kim (xem đoạn mã giả) khá giống với leo đồi. Thay vì chọn bước *tốt nhất*, nó chọn một bước *ngẫu nhiên*. Nếu bước di chuyển cải thiện tình hình thì nó luôn được chấp nhận. Ngược lại, thuật toán chấp nhận bước di chuyển với xác suất nhỏ hơn 1. Xác suất giảm dần theo hàm mũ trên "độ xấu" của bước di chuyển - sự lượng giá giảm một lượng ΔE. Xác suất cũng giảm khi "nhiệt độ" T giảm: "bước đi tồi" gần như được cho phép lúc đầu khi nhiệt độ còn cao và chúng khó còn như vậy khi T giảm. Ta có thể chứng minh rằng nếu lên lịch cho T giảm đủ chậm thì thuật toán sẽ tìm ra tối ưu toàn cục với xác suất tiến gần tới 1.

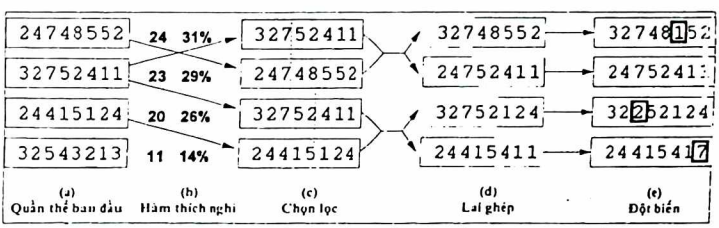
Mô phỏng luyện kim được sử dụng rộng rãi lần đầu để giải bài toán thiết kế vi mạch trong những năm đầu thập niên 80. Nó đã được áp dụng rộng rãi trong việc lập lịch sản xuất trong nhà máy và những công việc tối ưu qui mô lớn khác.

III.4 Thuật giải di truyền

Thuật giải di truyền (gọi tắt GA) là một biến thể của tìm kiếm nhóm ngẫu nhiên với trạng thái con được sinh ra bởi sự kết hợp của hai trạng thái cha mẹ, thay vì chỉ biến đổi một trạng thái. Sự tương đồng với chọn lọc tự nhiên cũng giống như trong tìm kiếm nhóm ngẫu nhiên, trừ việc bây giờ ta sẽ làm việc với sinh sản hữu tính thay vì sinh sản vô tính.

Giống như tìm kiếm nhóm, GA bắt đầu với tập k trạng thái phát sinh ngẫu nhiên, được gọi là **quần thể**. Mỗi trạng thái (hoặc cá thể) được thể hiện như một chuỗi kí tự hữu hạn, thông dụng nhất là chuỗi các số 0 và 1. Ví dụ, một trạng thái trong bài toán 8 hậu xác định vị trí của 8 quân hậu, mỗi cột 8 ô sẽ có một quân và như vậy cần $8 \times \log_2 8 = 24$ bit.

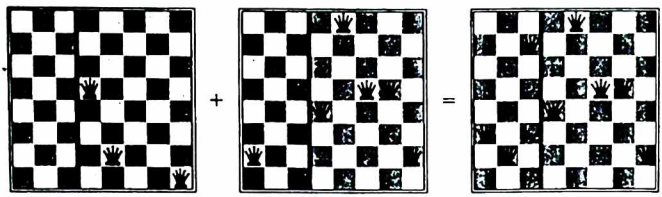
Một cách khác, trạng thái có thể được thể hiện bằng 8 con số, mỗi con số có giá trị từ 1 đến 8. Hình 2.10(a) cho thấy một quần thể gồm bốn chuỗi 8 kí số thể hiện các trạng thái của 8 quân hậu.



Hình 2.10 - *Thuật giải di truyền. Quần thể ban đầu ở* (a) *được xếp hạng theo hàm thích nghi ở* (b), *các cặp được chọn* (c). *Thế hệ con được tạo ra trong* (d) *và được đột biến trong* (e).

Việc sản sinh các trạng thái trong thế hệ tiếp theo để thể hiện trong Hình 2.10(b)-(e). Trong (b), mỗi trạng thái được đánh giá bởi hàm đánh giá hoặc (theo thuật ngữ của GA) hàm thích nghi. Hàm thích nghi trả về giá trị lớn hơn cho những trạng thái tốt hơn, nên với bài toán 8 hậu ta sử dụng số lượng các cặp hậu không tấn công nhau, khi đó lời giải sẽ có giá trị là 28. Giá trị cho bốn trạng thái trong hình là 24, 23, 20 và 11. Trong trường hợp cụ thể này, xác suất cá thể được

chọn để lai (sản sinh ra thế hệ sau) tỉ lệ trực tiếp với độ thích nghi của cá thể đó và con số phần trăm được ghi kế bên độ thích nghi.



Hình 2.11 - *Trạng thái tương ứng với hai cá thể cha mẹ đầu tiên trong Hình 2.9*(c) *và cá thể con trong Hình 2.9*(d). *Những cột bị tô mờ sẽ bị mất đi khi thực hiện phép lai và những cột còn lại sẽ được giữ nguyên.*

Trong (c), hai cặp được chọn ngẫu nhiên đem **lai ghép** với nhau. Lưu ý rằng một cá thể có thể được chọn một hay hai lần hoặc không được chọn. Với mỗi cặp đem lai này, vị trí lai trên chuỗi được chọn ngẫu nhiên. Trong Hình 2.10 vị trí lai từ sau số thứ ba cho cặp đầu tiên và sau số thứ năm cho cặp thứ hai.

Trong (d), các cá thể con được tạo ra bằng cách hoán đổi chuỗi của hai cha mẹ tại vị trí lai. Ví dụ, cá thể con đầu tiên của cặp đầu tiên sẽ lấy ba kí số đầu từ cá thể cha mẹ đầu tiên và những kí số còn lại lấy từ cá thể cha mẹ thứ hai, trong khi cá thể con thứ hai lấy ba kí số đầu từ cá thể cha mẹ thứ hai và phần còn lại từ cá thể cha mẹ đầu tiên. Trạng thái tham gia trong bước sản sinh thế hệ mới được thể hiện trong Hình 2.10. Ví dụ này thể hiện một điều rằng khi hai trạng thái cha mẹ rất khác nhau thì phép lai có thể tạo ra một trạng thái khác xa so với cả hai trạng thái cha mẹ. Thường thì trường hợp mà quần thể khá khác biệt trong giai đoạn đầu của quá trình, thì phép lai (giống như mô phỏng luyện kim) thường lúc đầu sẽ nhảy những bước lớn trong không gian trạng thái trong quá trình tìm kiếm và những bước nhỏ hơn trong giai đoạn về sau, khi hầu hết các cá thể đều khá giống nhau.

Cuối cùng, trong (e) mỗi vị trí được chọn để thực hiện **đột biến** ngẫu nhiên với một xác suất nhỏ độc lập. Một kí số được đột biến trong cá thể con đầu tiên, thứ ba và thứ tư. Trong bài toán 8 hậu, điều này tương đương với việc chọn ngẫu nhiên một quân hậu và di chuyển nó

```
function Thuật_giải_di_truyền (quần_thể, Hàm_Thích_nghi) return một cá thể
    input: quần_thể, tập các cá thể
           Hàm_Thích_nghi, hàm đo độ thích nghi của từng cá thể
    repeat
      quần_thể_mới←tập rỗng
      loop for i from 1 to Kích_thước(quần_thể) do
           x ←Lựa_chọn_Ngẫu_nhiên(quần_thể, Hàm_Thích_nghi)
           y ←Lựa_chọn_Ngẫu_nhiên(quần_thể, Hàm_Thích_nghi)
           con←Lai_ghép(x, y)
           if (xác suất ngẫu nhiên nhỏ) then con ←Đột_biến(con)
                 thêm con vàoquần_thể_mới
           quần_thể←quần_thể_mới
    until một số cá thể đủ phù hợp hoặc hết thời gian cho phép
    return cá thể tốt nhất trong quần_thể, dựa trên Hàm_Thích_nghi
function Lai_ghép(x, y)returns một cá thể
    input: x, y, cá thể cha mẹ
    n ←Chiều_dài(x)
    c ←phát sinh ngẫu nhiên một số từ 1 đến n
    return Thêm(Chuỗi_con(x, 1, c), Chuỗi_con(y, c + 1, n))
```

Hình 2.12-*Thuật giải di truyền.*

tới một ô ngẫu nhiên trong cùng một cột. Phần sau mô tả thuật toán cài đặt tất cả các bước trên.

Giống như tìm kiếm nhóm ngẫu nhiên, thuật giải di truyền kết hợp việc leo đồi với việc duyệt ngẫu nhiên và trao đổi thông tin giữa các hướng tìm kiếm song song. Ưu điểm chính (nếu có) của thuật giải di truyền có được nhờ phép lai. Nhưng ta có thể chứng minh toán học rằng nếu vị trí trên mã gen ban đầu được hoán đổi theo thứ tự ngẫu nhiên thì phép lai không mang đến ưu điểm gì. Theo trực giác, ta thấy ưu điểm đến từ khả năng lai để kết hợp những nhóm kí tự lớn đang tiến hóa riêng lẻ để tạo ra một hữu ích, do đó làm tăng mức độ mịn của công việc tìm kiếm. Ví dụ, có thể đặt ba quân hậu đầu ở vị trí 2, 4 và 6 (nơi mà chúng không tấn công lẫn nhau) tạo thành một nhóm hữu ích có thể kết hợp với những nhóm khác để cho ra lời giải.

Lý thuyết về thuật giải di truyền giải thích cách thức hoạt động của điều này sử dụng ý tưởng của lược đồ. **Lược đồ** là một chuỗi con trong đó một vài vị trí bị bỏ trống. Ví dụ, sơ đồ 246***** mô tả tất cả

các trạng thái của 8 hậu mà trong đó ba quân hậu đầu tiên lần lượt ở các vị trí 2, 4 và 6. Những chuỗi khớp với sơ đồ (24613578 chẳng hạn) được gọi là các **thể hiện** của sơ đồ đó. Ta có thể chứng minh rằng, số lượng thể hiện của sơ đồ trong quần thể sẽ tăng theo thời gian. Thuật giải di truyền thực hiện tốt khi các sơ đồ tương ứng với các thành phần có nghĩa trong lời giải. Điều này cho thấy yếu tố thành công của giải thuật di truyền đòi hỏi sự cẩn thận trong thiết kế thể hiện.

Trong thực tế, thuật giải di truyền có sự ảnh hưởng rộng lớn đến các bài toán tối ưu hóa, chẳng hạn thiết kế vi mạch và lập lịch công việc. Nhiều việc cần phải làm để xác định điều kiện để giải thuật di truyền đạt hiệu quả cao.

IV. KẾT CHƯƠNG

Chương này đã khảo sát việc ứng dụng của các *heuristic* để giảm chi phí tìm kiếm.

- Tìm kiếm tối ưu chỉ là tìm kiếm trên đồ thị trong đó các nút chưa mở có chi phí tối thiểu (dựa trên một độ đo nào đó) được lựa chọn để mở rộng. Các thuật toán tối ưu thường dùng một hàm *heuristic h(n)* để ước lượng chi phí từ nút *n*.
- Tìm kiếm tối ưu kiểu tham lam mở các nút có *h(n)* nhỏ nhất. Thuật toán không đạt được tối ưu nhưng thường hiệu quả.
- Tìm kiếm A* mở các nút có $f(n)=g(n)+h(n)$ nhỏ nhất. A* đầy đủ và tối ưu nếu hàm *h(n)* là chấp nhận được. Độ phức tạp về mặt không gian của A* vẫn rất cao.
- Hiệu quả của các thuật toán tìm kiếm *heuristic* phụ thuộc vào chất lượng của các hàm *heuristic*. Các *heuristic* tốt có thể được xây dựng bằng cách giản lược định nghĩa bài toán, hoặc bằng cách tính toán trước các chi phí cho các bài toán con trong cơ sở dữ liệu mẫu.
- Các phương pháp tìm kiếm cục bộ như leo đồi hoạt động trên các công thức trạng thái đầy đủ, chỉ cần giữ một số lượng nhỏ các nút trong bộ nhớ. Một số thuật toán ngẫu nhiên cũng đã

được phát triển, bao gồm luyện thép, cho ra các lời giải tối ưu khi được cung cấp các tham số thích hợp. Nhiều phương pháp tìm kiếm cục bộ cũng có thể được dùng để giải các bài toán trong không gian liên tục.

- Thuật giải di truyền là một thuật giải tìm kiếm leo đồi ngẫu nhiên, trong đó một quần thể lớn các trạng thái được duy trì. Các trạng thái mới có thể được phát sinh bằng cách đột biến và lai ghép.

V. THUẬT NGỮ TIẾNG ANH

Sau đây là những thuật ngữ tiếng Anh liên quan đến những khái niệm đã dùng ở chương này:

admissible heurestic: chấp nhận được

best-first search: tìm kiếm tối ưu

crossover: lai ghép

evaluation function: hàm đánh giá

first-choice hill climbing: leo đồi có lựa chọn

fitness function: hàm thích nghi

genetic algorithm: thuật giải di truyền

global maximum: cực đại toàn cục

global minimum: cực tiểu toàn cục

greedy best-first search: tìm kiếm tối ưu kiểu tham lam, tìm kiếm tham lam

greedy local search: tìm kiếm cục bộ tham lam

hill-climbing search: tìm kiếm leo đồi

imformed search: tìm kiếm có thông tin

instance: thể hiện

local beam search: tìm kiếm nhóm cục bộ

local search: tìm kiếm cục bộ

mutation: đột biến

objective function: hàm mục tiêu

optimization problem: bài toán tối ưu

plateau: mặt bằng phẳng

population: quần thể

random-restart hill climbing: leo đồi làm lại ngẫu nhiên

relaxed problem: bài toán giản lược

ridge: chóp

schema: lược đồ

state space landscape: địa hình không gian trạng thái

stimulated annealing: tìm kiếm mô phỏng luyện kim

stochastic hill climbing: leo đồi ngẫu nhiên

subproblem: bài toán con

Chương 3
BÀI TOÁN THỎA MÃN RÀNG BUỘC

Trong chương 1 và 2, chúng ta đã bước đầu làm quen với các ý tưởng trí tuệ nhân tạo với việc áp dụng các tri thức bổ sung vào các thuật toán tìm kiếm nhằm nâng cao hiệu quả. Chương này sẽ xem xét một trong những dạng đặc biệt của bài toán tìm kiếm là bài toán thỏa mãn ràng buộc. Phần I giới thiệu định nghĩa và một số ví dụ cụ thể của bài toán. Thuật toán quay lui tổng quát giải bài toán thỏa mãn ràng buộc được giới thiệu trong phần II. Một số *heuristic* để giải bài toán nhằm tìm kiếm những lời giải hợp lý sẽ được đưa ra trong phần III.

I. BÀI TOÁN THỎA MÃN RÀNG BUỘC (CSP)

Bài toán thỏa mãn ràng buộc (CSP) được định nghĩa bởi một tập các **biến:** $X_1, X_2,..., X_n$ và một tập các **ràng buộc:** $C_1, C_2,..., C_m$. Mỗi biến X_i có **miền xác định** D_i khác rỗng. Mỗi ràng buộc C_i bao gồm một số tập con các biến và các mối liên hệ giữa các tập con này. Một trạng thái của bài toán được định nghĩa bởi **phép gán** giá trị đến một số hay tất cả các biến, $\{X_i = v_i, X_j = v_j,...\}$. Một phép gán không vi phạm bất kì ràng buộc nào được gọi là phép gán **hợp lệ**. Một phép gán đầy đủ là một phép gán trong đó mỗi biến đều được gán và **lời giải** của CSP là một phép gán đầy đủ thỏa mãn tất cả các ràng buộc. Một số CSP cũng đòi hỏi một lời giải làm cực đại hóa **hàm mục tiêu.**

Một ví dụ cụ thể của bài toán thỏa mãn ràng buộc là bài toán tô màu bản đồ nước Úc. Một bản đồ của nước Úc bao gồm các bang và vùng lãnh thổ được biểu diễn trong Hình 3.1 (a). Yêu cầu của bài toán cần tô màu cho mỗi miền hoặc màu đỏ, xanh lá cây, hay xanh dương trong đó không có miền nào kề nhau được tô chung một màu. Để đưa bài

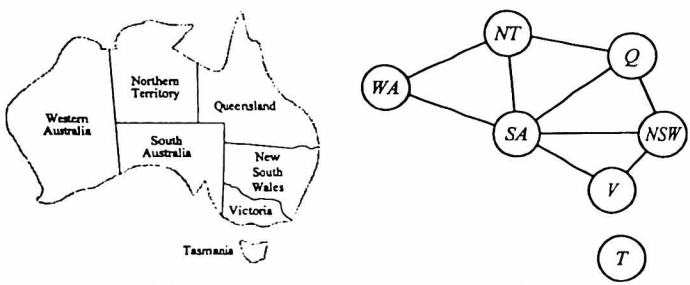
toán về dạng CSP, chúng ta định nghĩa các biến là các miền: WA, NT, Q, NSW, V, SA và T. Miền giá trị của mỗi biến là tập: {đỏ, xanh lá, xanh dương}. Ràng buộc đòi hỏi các miền kề nhau có màu phân biệt, ví dụ quan hệ hợp lệ cho WA và NT là các cặp:

{(đỏ, xanh lá), (đỏ, xanh dương), (xanh lá, đỏ), (xanh lá, xanh dương), (xanh dương, đỏ), (xanh dương, xanh lá)}.

Chúng ta có rất nhiều lời giải như:

{WA = đỏ, NT = xanh lá, Q = đỏ, NSW = xanh lá, V= đỏ, SA= xanh dương, T = đỏ}.

Ta có thể xem xét một bài toán thoả mãn ràng buộc dưới dạng một đồ thị ràng buộc, như trong Hình 3.1(b). Các nút của đồ thị tương ứng với các biến của bài toán và cung nối tương ứng với ràng buộc.



(a) (b)

Hình 3.1 - (a). *Những bang chính của Úc. Tô màu bản đồ này có thể được xem như bài toán thỏa mãn ràng buộc. Mục tiêu có thể gán màu cho mỗi miền để mà không có miền nào kề nhau được tô cùng một màu.* (b). *Bài toán tô mầu được thể hiện bằng một đồ thị ràng buộc.*

Một CSP có thể được tiến hành theo một công thức của bài toán tìm kiếm thông thường như sau:

- **Trạng thái ban đầu**: Tập gán rỗng {}, tất cả các biến đều chưa được gán.
- **Hàm trạng thái con**: Một giá trị có thể được gán đến bất kì biến nào, giả sử rằng không ảnh hưởng đến các biến đã được gán trước đó.

- **Trạng thái đích**: Việc gán hiện tại được hoàn thành.
- **Chi phí đường đi**: Một chi phí hằng số cho mỗi bước.

Mỗi lời giải phải là một phép gán đầy đủ và nó sẽ xuất hiện ở độ sâu n nếu có n biến. Cây tìm kiếm mở rộng chỉ đến độ sâu n. Vì vậy, thuật toán tìm kiếm theo chiều sau được áp dụng phổ biến cho CSP. Cũng có trường hợp đường đi cho ra lời giải không đúng. Từ đây về sau, chúng ta có thể sử dụng một **công thức trạng thái đầy đủ** trong đó mỗi trạng thái là một phép gán đầy đủ dù có hay không thỏa mãn những ràng buộc. Phương pháp tìm kiếm cục bộ làm việc tốt với công thức này.

Bài toán CSP đơn giản nhất bao gồm các biến **rời rạc** và có **miền xác định hữu hạn**. Bài toán tô màu bản đồ thuộc về dạng này. Bài toán 8 hậu cũng có thể được xem như là CSP miền xác định hữu hạn, ở đó các biến $Q_1,\ldots, Q_8$ là vị trí của mỗi quân hậu trong các cột $1,\ldots,8$ và mỗi biến có miền xác định $\{1, 2, 3, 4, 5, 6, 7, 8\}$. Nếu miền kích thước lớn nhất của bất kì biến nào trong CSP là d, thì số lượng phép gán đầy đủ có thể là $O(d^n)$ – số mũ là số lượng biến. CSP miền xác định hữu hạn cũng bao gồm **CSP luận lý**, giá trị các biến là đúng hoặc sai. CSP luận lý cũng là trường hợp đặc biệt của bài toán NP - đầy đủ như bài toán *3-SAT* (Kiểm tra xem một biểu thức logic 3 biến có thỏa hay không). Do đó, trong trường hợp xấu, chúng ta không thể giải quyết CSP miền xác định hữu hạn trong thời gian ít hơn hàm mũ.

Các biến rời rạc cũng có **miền xác định vô hạn** – ví dụ, tập những số tự nhiên hay tập những chuỗi kí tự. Một ví dụ khác, khi lập lịch công việc xây dựng, ngày bắt đầu của mỗi công việc là một biến và giá trị khả dĩ là số nguyên những ngày tính từ ngày hiện tại. Với miền xác định vô hạn, ta không còn mô tả các ràng buộc bằng cách liệt kê tất cả những liên hệ cho phép giữa những giá trị mà thay vào đó sử dụng **ngôn ngữ ràng buộc**. Ví dụ, nếu CV_1 mất 5 ngày, phải trước CV_3 thì chúng ta cần ngôn ngữ ràng buộc bất đẳng thức đại số như $BắtĐầuCV_1 + 5 \leq BắtĐầuCV_3$.

Bên cạnh việc kiểm tra loại biến có thể xuất hiện trong CSP, chúng ta cũng sẽ xem xét loại ràng buộc. Loại đơn giản nhất là **ràng buộc bậc nhất**, ở đó nó ràng buộc giá trị của từng biến đơn. Ví dụ, người miền

nam nước Úc không thích màu xanh lá. Mỗi ràng buộc đơn có thể bỏ đi đơn giản bằng cách tiền xử lý miền biến tương ứng để bỏ đi các giá trị mà vi phạm ràng buộc. Một **ràng buộc bậc hai** liên quan đến hai biến. Ví dụ, SA ≠ NSW là một ràng buộc kép. Một CSP kép là một bài toán với chỉ các ràng buộc kép, nó có thể biểu diễn bằng một đồ thị ràng buộc như hình 3.1(b).

Ràng buộc bậc cao hơn bao gồm 3 hay nhiều biến trở lên. Một ví dụ quen thuộc đó là bài toán đố (xem hình 3.2(a)). Thông thường chúng ta ràng buộc mỗi kí tự đại diện cho một chữ số phân biệt. Trường hợp ở hình 3.2(a), nó thể hiện ràng buộc 6 biến *TấtCảKhácNhau(F, T, U, W, R, O)*. Tương tự, nó có thể được thể hiện bằng một tập các ràng buộc kép như F ≠ T. Những ràng buộc thêm trên 4 cột của bài toán cũng liên quan đến các biến và có thể được viết như sau:

$$O + O = R + 10 \times X_1.$$

$$X_1 + W + W = U + 10 \times X_2.$$

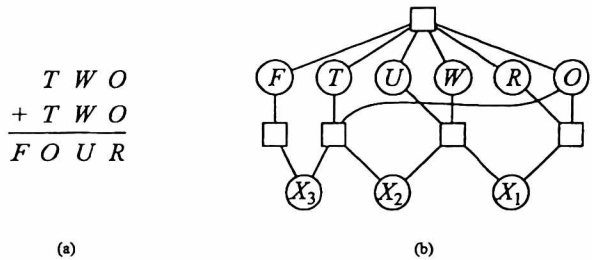
$$X_2 + T + T = O + 10 \times X_3.$$

$$X_3 = F.$$

Trong đó: X_1, X_2, X_3 là các biến phụ thể hiện chữ số (0 hoặc 1) cần mang qua cột kế tiếp. Ràng buộc bậc cao có thể được biểu diễn trong một siêu đồ thị như hình 3.2(b). Trong trường hợp này, ràng buộc *TấtCảKhácNhau* có thể bị phân nhỏ thành các ràng buộc kép - F≠T, F≠U,...Thực tế, mỗi ràng buộc không gian hữu hạn bậc cao có thể được biến đổi thành một tập các ràng buộc kép nếu đủ các biến tạm như đã đề cập ở ví dụ trên. Vì vậy, chương này chỉ quan tâm đến các ràng buộc kép.

Tất cả ràng buộc đã được mô tả đến bây giờ đều là các ràng buộc tuyệt đối, vi phạm những luật này sẽ không dẫn đến lời giải tiềm năng. Rất nhiều CSP thực tế bao gồm các ràng buộc sở thích chỉ ra lời giải nào sẽ tốt hơn. Ví dụ, trong bài toán lập thời gian biểu ở đại học, giáo sư X thích dạy buổi sáng, ngược lại giáo sư Y thích dạy buổi chiều. Một thời gian biểu sắp giáo sư X dạy lúc 2 giờ chiều sẽ là một lời giải nhưng không là một lời giải tối ưu. Ràng buộc sở thích có thể được xem như là chi phí gán cho một biến cá thể – chẳng hạn, gán một chỗ

buổi chiều cho giáo sư X làm mất 2 điểm của hàm mục tiêu toàn bộ, ngược lại vào chỗ buổi sáng chỉ mất 1. Với công thức này, CSP với sở thích có thể được giải quyết sử dụng phương pháp tìm kiếm tối ưu, hoặc dựa trên đường đi hoặc cục bộ.



Hình 3.2 - (a). *Bài toán đố tìm ẩn số. Mỗi kí tự đại diện cho một chữ số phân biệt; mục tiêu là tìm ra một thay thế của những chữ số cho những kí tự để mà tổng kết quả là một con số đúng, với giới hạn cộng là không dẫn đến 0 được cho phép.* (b). *Siêu đồ thị ràng buộc cho bài toán đi tìm ẩn số, cho thấy ràng buộc TấtCảKhácNhau bên cạnh các ràng buộc cộng từng cột. Mỗi ràng buộc là một kết nối hình ô vuông đến các biến mà nó ràng buộc.*

II. TÌM KIẾM QUAY LUI CHO CSP

Phần trước giới thiệu công thức của CSP dưới dạng bài toán tìm kiếm. Sử dụng công thức này, bất kì thuật toán tìm kiếm nào từ chương trước đều có thể giải quyết CSP. Giả sử chúng ta áp dụng tìm kiếm theo chiều sâu lên công thức của một họ các bài toán CSP. Một số vấn đề có thể được lưu ý là: hệ số phân nhánh ở mức đầu là *nd*, bởi vì bất kì những giá trị *d* có thể được gán đến bất kì *n* biến nào. Ở mức kế tiếp, hệ số phân nhánh là *(n-1)d* và cứ thế cho đến mức *n*. Chúng ta sẽ tạo ra một cây với *n!dn* nút lá, mặc dù chỉ có *dn* phép gán đầy đủ.

Lí do của vấn đề này là do chúng ta đã bỏ qua một thuộc tính quan trọng của tất cả các CSP: **tính giao hoán**. Một bài toán có tính giao hoán nếu thứ tự áp dụng bất kì tập hành động nào cho trước không

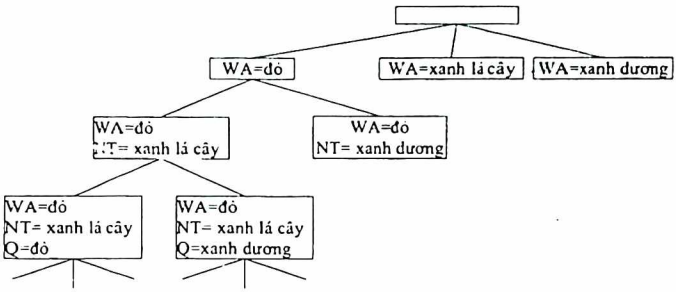
ảnh hưởng tới kết quả thu được. Đây là trường hợp xảy ra với CSP bởi vì khi gán giá trị cho các biến, chúng ta gán từng phần giống nhau mà không quan tâm đến thứ tự. Vì vậy, tất cả các thuật toán tìm kiếm CSP phát sinh trạng thái con bằng cách xem xét gán giá trị chọ chỉ một biến đơn ở mỗi nút trong cây tìm kiếm. Ví dụ, tại nút gốc của một cây tìm kiếm cho bài toán tô màu nước Úc, chúng ta phải có một chọn lựa giữa: *SA = đỏ*, *SA = xanh lá*, và *SA = xanh dương*, nhưng chúng ta sẽ không bao giờ chọn lựa giữa: *SA = đỏ* và *WA = xanh dương*. Với giới hạn này, số nút lá là d^n.

Tìm kiếm quay lui được sử dụng trong thuật toán tìm kiếm chiều sâu bằng cách chọn các giá trị cho một biến cùng một lúc và quay lui khi một biến không có giá trị hợp lệ để gán. Bởi vì thể hiện của CSP đã được chuẩn hóa nên không cần thiết hỗ trợ tìm kiếm quay lui với một trạng thái khởi tạo xác định, hàm con cháu hay kiểm chứng lời giải. Một phần của cây tìm kiếm cho bài toán bản đồ Úc được minh họa trong Hình 3.4, ở đó chúng ta có các biến được gán theo thứ tự WA, NT, Q, ...

```
function Tìm_kiếm_Quay_lui(csp) return lời giải hoặc thất bại
return Quay_lui_Đệ_qui({},csp)

functionQuay_lui_Đệ_qui(tập_gán,csp) return một lời giải hoặc thất bại
    if tập_gán đầy đủ then return tập_gán
    biến←Chọn_Biến_Chưa_Gán(các_biến[csp], tập_gán, csp)
    for each giá_trị in miền_giá_trị_có_thứ_tự(tập_gán, csp)
        thêm {biến=giá_trị} vào tập_gán
        kết_quả ←Quay_lui_Đệ_qui(tập_gán,csp)
        if kết_quả ≠ thất bại then return kết_quả
        loại bỏ {biến=giá_trị} khỏi tập_gán
    return thấtbại
```

Hình 3.3- *Thuật toán Tìm kiếm Quay lui.*



Hình 3.4 - *Một phần của cây tìm kiếm được phát sinh bởi thuật toán quay lui đơn giản đối với bài toán tô màu bản đồ trong Hình 3.1.*

III. TÌM KIẾM HEURISTIC CHO CSP

Thuật toán quay lui truyền thống là một thuật toán không có thông tin, vì vậy rất kém hiệu quả cho các bài toán lớn. Trong những chương trước, chúng ta đã khắc phục điểm yếu của thuật toán tìm kiếm không có thông tin bằng cách thêm những hàm *heuristic* trên miền xác định rút ra từ tri thức của bài toán. Trong bài toán CSP, ta có thể định nghĩa nhiều *heuristic* khác nhau. Thuật toán quay lui có dòng lệnh:

biến ← Chọn_Biến_Chưa_Gán(*các_biến*[*csp*], *tập_gán*, *csp*).

Theo mặc định, Chọn_Biến_Chưa_Gán đơn giản chọn những biến chưa được gán kế tiếp theo thứ tự được định bởi danh sách *các_biến*[*csp*]. Thứ tự biến tĩnh này hiếm khi mang lại kết quả tìm kiếm hiệu quả nhất. Ví dụ, sau khi gán cho *WA = đỏ*, *NT = xanh lá*, chỉ còn một giá trị có thể cho *SA*, vì vậy *SA = xanh dương*. Thực tế, sau khi *SA* được gán, chọn lựa *Q*, *NSW* và *V* là bắt buộc. Ý tưởng này – chọn biến với ít giá trị hợp lệ nhất – được gọi là *heuristic* **giá trị còn lại tối tiểu (MRV)**. Biến này cũng được gọi là "biến được ràng buộc nhất" hay *heuristic* "theo thất bại", bởi vì thuật giải chọn một biến mà gần trường hợp gây ra lỗi sớm, do đó có thể giảm bớt nhánh cho cây tìm kiếm. Nếu có một biến X không còn giá trị hợp lệ, *heuristic* MRV

sẽ chọn X và thất bại sẽ được phát hiện ngay lập tức – tránh việc tìm kiếm điểm các biến khác và luôn luôn thất bại khi X cuối cùng cũng được chọn. Cách này thực thi tốt hơn từ 3 đến 3.000 lần so với quay lui đơn giản phụ thuộc vào từng bài toán cụ thể.

Heuristic MRV không phải lúc nào cũng hữu ích trong việc chọn vùng đầu tiên nước Úc để tô màu, bởi vì khởi tạo mỗi vùng đều có 3 màu hợp lệ để tô. Trong trường hợp này, ***heuristic* theo bậc** thường được chọn. Nó cố gắng giảm hệ số phân nhánh trong chọn lựa ở tương lai bằng cách chọn biến liên quan đến số lượng các ràng buộc lớn nhất từ những biến chưa được gán trước đó. Trong Hình 3.1, *SA* là biến có bậc cao nhất bằng 5, những biến khác có bậc 2 hoặc 3, ngoại trừ *T*, có bậc 0. Thực tế, một khi *SA* được chọn, áp dụng *heuristic* bậc để giải quyết bài toán mà không cần bất kì bước nào sai – có thể chọn bất kì màu phù hợp ở mỗi điểm và vẫn đạt đến một lời giải mà không cần quay lui.

Một khi một biến được chọn, thuật toán phải quyết định thứ tự để kiểm tra giá trị của nó. Để làm được điều này, ***heuristic* giá trị ràng buộc ít nhất** có thể giúp ích trong một số trường hợp. Nó chọn giá trị ngăn cản ít nhất sự chọn lựa của các biến láng giềng trong đồ thị ràng buộc. Ví dụ, giả sử trong Hình 3.1, chúng ta đã gán giá trị từng phần với *WA* = *đỏ* và *NT* = *xanh lá*, bước chọn kế tiếp là *Q*. Màu *xanh dương* là chọn lựa không tốt bởi vì nó đã bỏ đi giá trị hợp lệ cuối cùng dành cho láng giềng của *Q*, *SA*. Vì vậy, *heuristic* giá trị ràng buộc ít nhất sẽ chọn màu *đỏ* thay vì *xanh dương*. Nhìn chung, *heuristic* cố gắng để càng nhiều lựa chọn càng tốt cho các phép gán biến phía sau. Tất nhiên, nếu chúng ta muốn tìm tất cả các giải pháp có thể cho bài toán, thì thứ tự làm không còn quan trọng nữa bởi vì chúng ta phải xem xét từng giá trị một. Điều tương tự cũng xảy ra nếu bài toán không có lời giải.

IV. KẾT CHƯƠNG

Chương này giới thiệu các định nghĩa cơ bản của một bài toán thoả mãn ràng buộc. Các kiến thức chuyên sâu đối với bài toán này sẽ được giới thiệu trong các chương trình cao hơn. Một số điểm cơ bản có thể được rút ra như sau:

- Các bài toán thoả mãn ràng buộc (CSP) bao gồm các biến với các ràng buộc. Nhiều bài toán thực tế quan trọng có thể được mô tả thành các bài toán thoả mãn ràng buộc. Cấu trúc của một CSP có thể được biểu diễn thành một đồ thị ràng buộc.
- Tìm kiếm quay lui, là một dạng của tìm kiếm theo chiều sâu, thường được dùng để giải quyết các CSP.
- Các *heuristic* giá trị còn lại tối tiểu và *heuristic* theo bậc là các phương pháp không phụ thuộc vào từng bài toán cụ thể và thường được sử dụng để làm giảm đáng kể hệ số phân nhánh của bài toán.

V. THUẬT NGỮ TIẾNG ANH

Sau đây là những thuật ngữ tiếng Anh liên quan đến những khái niệm đã dùng ở chương này:

absolute constraint: ràng buộc tuyệt đối

assignment: phép gán

backtracking search: tìm kiếm quay lui

binary constraint: ràng buộc bậc hai

Boolean CSP: CSP luận lý

commutativity: tính giao hoán

complete-state formulation: công thức trạng thái đầy đủ

consistent: hợp lệ

constraint: ràng buộc

constraint language: ngôn ngữ ràng buộc

constraint satisfaction problem (CSP): bài toán thoả mãn ràng buộc

degree heuristic: heuristic theo bậc

discrete: rời rạc

domain: miền xác định

finite domain: miền xác định hữu hạn

hypergraph: siêu đồ thị

infinite domain: miền xác định vô hạn

least constraing value: giá trị còn lại ít nhất

minimum remaining value (MRV): giá trị còn lại tối tiểu

objective function: hàm mục tiêu

preference constraint: ràng buộc sở thích

solution: lời giải

unary constraint: ràng buộc bậc nhất

variable: biến

Phần 2

BIỂU DIỄN TRI THỨC VÀ SUY DIỄN

Chương 4
LOGIC VÀ SUY DIỄN

Phần 1 của giáo trình đã tập trung vào giải quyết những câu hỏi làm thế nào để xây dựng quá trình tìm kiếm và bỏ qua những câu hỏi về những phương pháp tổng quát nhằm mô tả các trạng thái và hành động trong bài toán. Tuy nhiên các hệ thống giao tiếp với con người đôi khi đòi hỏi cao hơn việc chỉ đưa ra lời giải cuối cùng. Người dùng còn muốn hiểu được lời giải một cách tường minh. Phần 2 của giáo trình bổ sung thêm cho hệ thống khả năng suy diễn logic. Một hệ thống dựa vào tri thức và logic thường khởi đầu bằng một số tri thức về thế giới xung quanh cũng như tri thức hành động của nó. Hệ thống sử dụng suy diễn logic nhằm duy trì mô tả về thế giới nhằm đạt tới những đối tượng tri giác mới và suy luận để nhằm đi đến mục đích mong muốn.

Chương 4 sẽ giới thiệu một số phác họa cơ bản cách xây dựng một hệ thống dựa vào cơ sở tri thức. Chương này cũng trình bày một ngôn ngữ logic đơn giản giúp biểu diễn tri thức và sẽ hướng dẫn cách sử dụng những tri thức để đưa ra những kết luận về thế giới và quyết định nên làm gì. Chương 5 và 6 sẽ bổ sung thêm ngôn ngữ nhằm giúp hệ thống có khả năng biểu diễn những trạng thái mở rộng của tri thức trong thế giới phức tạp. Chương 7 giới thiệu một số kỹ thuật để giải quyết vấn đề không chắc chắn trong tri thức.

1. LOGIC

Về mặt hình thức, logic bao gồm những thành phần sau:

1. Hệ thống hình thức để mô tả trạng thái sự vật, bao gồm:
   a. **Cú pháp** ngôn ngữ, mô tả làm thế nào tạo nên các câu.
   b. **Ngữ nghĩa** ngôn ngữ, xác định các ràng buộc ngữ nghĩa về mối quan hệ giữa các trạng thái sự vật.

2. **Lý thuyết chứng minh** – tập các luật cho suy diễn những đặc tính kế thừa của các câu.

Giáo trình sẽ giới thiệu hai loại logic: logic mệnh đề hay logic boolean và logic bậc nhất. Trong logic mệnh đề, các kí hiệu biểu diễn hầu hết các mệnh đề (các thành phần của suy luận); ví dụ, *D* có cách hiểu là "*Aristotle* chết" có thể là mệnh đề đúng hoặc không đúng. Các kí hiệu mệnh đề có thể được nối với nhau sử dụng các phép toán logic để tạo ra các câu có ý nghĩa phức tạp hơn. Như vậy logic mệnh đề tạo ra rất ít thông tin để biết các vấn đề được biểu diễn thế nào.

Logic bậc nhất tuân theo các biểu diễn của thế giới theo các thuật ngữ của đối tượng hoặc các biến vị từ trên đối tượng (ví dụ, các tính chất hoặc các mối quan hệ của đối tượng), cũng như sử dụng các phép kết hợp hoặc các phép lượng hóa, cho phép các câu có thể biểu diễn về mọi thứ trong thế giới và đã được nghiên cứu hàng trăm năm trước. Do đó, phần lớn nội dung của phần logic sẽ tập trung giới thiệu cách biểu diễn và sử dụng logic bậc nhất. Nhưng trước tiên, chúng ta sẽ làm quen với các vấn đề của logic qua loại đơn giản hơn: logic mệnh đề.

II. LOGIC MỆNH ĐỀ: LOẠI LOGIC ĐƠN GIẢN.

Mặc dù có những giới hạn trong việc biểu diễn, logic mệnh đề vẫn có vai trò minh họa nhiều định nghĩa logic giống như logic bậc nhất. Phần này sẽ mô tả cú pháp, ngữ nghĩa và các thủ tục suy luận liên quan của nó.

II.1 Cú pháp

Cú pháp của logic mệnh đề rất đơn giản. Các kí hiệu của logic mệnh đề là các hằng số logic *True* hoặc *False*, các kí hiệu mệnh đề như *P* và *Q*, liên kết logic $\wedge$, $\vee$, $\Leftrightarrow$, $\Rightarrow$ và các dấu ngoặc ( ). Tất cả các câu được tạo ra bằng cách liên kết các kí hiệu với nhau sử dụng theo các luật sau:

- Các hằng số logic *True* hoặc *False* là các câu.
- Một kí hiệu mệnh đề như *P* hoặc *Q* là các câu.
- Các dấu ngoặc bao quanh một câu tạo ra một câu, ví dụ, *(P A Q)*.

Một câu phức có thể được hình thành bằng cách liên kết từ các câu đơn giản hơn với 1 trong 5 phép liên kết logic:

- $\wedge$ ***(và)***: Một câu mà liên kết chính của nó là phép $\wedge$, như $P \wedge (Q \vee R)$, được gọi là phép giao; các thành phần của nó giao với nhau ($\wedge$ giống kí tự 'A' viết tắt của "*And*").
- $\vee$ ***(hoặc)***: Một câu sử dụng phép hoặc là hội, hội của các mệnh đề với nhau. (Là kí tự bắt đầu của từ Latinh "*Vel*" có nghĩa là *hoặc*).
- $\Rightarrow$ ***(kéo theo)***: Một câu kiểu như $(P \wedge Q) \Rightarrow R$ được gọi là phép kéo theo (hay điều kiện). $P \wedge Q$ là tiền đề và R là kết luận. Phép kéo theo cũng được gọi là các luật hoặc câu nếu thì. Kí hiệu phép kéo theo thỉnh thoảng được viết $\supset$ hoặc $\rightarrow$.
- $\Leftrightarrow$ ***(tương đương)***: Câu $(P \wedge Q) \Leftrightarrow (Q \wedge P)$ là tương đương.
- $\neg$ ***(phủ định)***: Một câu $\neg P$ được gọi là phủ định của P. Tất cả các toán tử liên kết, kết hợp 2 câu thành 1 câu; $\neg$ là phép toán liên kết duy nhất thao tác trên một toán tử đơn.

Ngữ pháp của logic là không rõ ràng – một câu như: $P \wedge Q \vee R$ có thể được phân tích là: $(P \wedge Q) \vee R$ hoặc $P \wedge (Q \vee R)$. Điều này tương đương với sự không rõ ràng của các phép toán số học như: $P + Q \times R$ và cách giải quyết sự tối nghĩa này cũng tương tự: sử dụng độ ưu tiên toán tử và sử dụng các dấu ngoặc bất cứ khi nào có thể xảy ra nhầm lẫn. Độ ưu tiên toán tử trong logic mệnh đề (từ cao đến thấp): $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$. Do vậy câu:

$\neg P \vee Q \wedge R \Rightarrow S$

Tương đương với câu sau:

$((\neg P) \vee (Q \wedge R)) \Rightarrow S$

III.2 Ngữ nghĩa

Ngữ nghĩa của logic mệnh đề cũng không phức tạp. Chúng ta định nghĩa nó bằng cách xác định cách hiểu của các kí hiệu mệnh đề, các hằng và nắm định ý nghĩa của các phép nối logic.

Một kí hiệu mệnh đề có thể mang bất cứ nghĩa nào. Cách hiểu của nó có thể bất cứ nghĩa gì. Cách hiểu của P có thể kiểu như là "Paris là thủ

đô của Pháp" hoặc "Aristotle bị chết". Một câu chứa chỉ một kí hiệu mệnh đề là thỏa mãn được nhưng không hợp lệ: nó chỉ đúng khi sự việc xảy ra đúng như thế.

Với các hằng số logic, một câu đúng luôn có một cách hiểu trong thế giới nó là sự kiện đúng và câu sai luôn luôn được hiểu là sai.

Một câu phức có ý nghĩa từ sự kết hợp của ý nghĩa các câu thành phần của nó. Mỗi liên kết có thể coi như là một hàm. Tương tự như phép cộng là một hàm với 2 đối số đầu vào và trả về một số, phép "*và*" là một hàm với 2 đối số đầu vào và trả về giá trị tương ứng trong bảng chân trị. Dưới đây là **bảng chân trị** của các phép toán logic.

Bảng 4-1. *Bảng chân trị của 5 phép liên hợp logic.*

P	Q	¬P	P ∧ Q	P ∨ Q	P ⇒ Q	P ⇔ Q
False	False	True	False	False	True	True
False	True	True	False	True	True	False
True	False	False	False	True	False	False
True	True	False	True	True	True	True

Bảng chân trị định nghĩa ngữ nghĩa của câu như *True* ∧ *True*. Câu phức $P \vee Q \wedge \neg S$, để xác định nghĩa của câu này ta xác định $P \vee Q$ và $\neg S$ trước, sau đó liên kết chúng bằng phép giao mệnh đề. Phương pháp tương tự như cách ta tính một biểu thức số học phức tạp.

Trong một số trường hợp, phép **kéo theo** ⇒ là quan trọng nhất, bảng chân trị của nó có vẻ như hơi khó hiểu, vì nó không giống với cách hiểu của chúng ta "*P kéo theo Q*" hoặc "*Nếu P thì Q*". Một điều nữa, logic mệnh đề không yêu cầu bất kì mối quan hệ nhân quả nào hoặc sự hợp lí nào giữa *P* và *Q*. Câu "*5 là số lẻ kéo theo Tokyo là thủ đô của Nhật Bản*" là câu đúng trong logic mệnh đề (theo cách hiểu thông thường), mặc dầu ta thấy câu này có vẻ kì lạ. Một điểm khác cũng dễ gây ra nhầm lẫn là phép kéo theo sẽ luôn đúng nếu mệnh đề phía trước là false. Ví dụ "*Nếu 5 là số chẵn thì Lâm thông minh*" là *true*, cho dù Lâm có thông minh hay không.

II.3 Tính tương đương, hợp lệ và thoả mãn được.

Một câu là hợp lệ nếu và chỉ nếu bất chấp cách hiểu trong thế giới như thế nào, nó mang ý nghĩa gì và trạng thái sự việc nó được dù được mô tả thế nào, thì nó vẫn đúng. Ví dụ, câu sau:

"Trời mưa hoặc trời không mưa"

luôn đúng bởi vì một trong 2 câu chắc chắn sẽ đúng bất chấp câu kia có đúng hay không. Ngược lại:

"Có một khoảng trống trong ô vuông trước mặt tôi hoặc có một bức tường trong ô vuông ở trước mặt tôi."

là không luôn đúng. Nó chỉ đúng nếu ta giả thiết rằng mỗi ô vuông hoặc chỉ có tường hoặc chỉ có khoảng trống trong nó. Vì vậy câu:

"Nếu một ô vuông chỉ chứa tường hoặc khoảng trống, thì có một khoảng trống trong ô vuông trước mặt tôi hoặc có một một bức tường trong ô vuông ở trước mặt tôi."

là hợp lệ.

Một câu là **thỏa mãn được** nếu và chỉ một số cách hiểu trong thế giới làm cho câu này đúng. Câu "*Trời mưa*" là thoả mãn được vì trời có thể mưa, mặc dầu lúc này trời không mưa. Các câu tự mâu thuẫn luôn là không thỏa mãn nếu sự mâu thuẫn không phụ thuộc lên nghĩa của những kí hiệu. Ví dụ, câu:

"Có bức tường ở trước mặt tôi và không có tường ở trước mặt tôi".

là một câu không thỏa mãn.

Bảng chân trị có thể được sử dụng không chỉ để xác định các phép liên kết, mà còn sử dụng để kiểm chứng các câu hợp lệ. Cho một câu, chúng ta tạo ra một bảng chân trị với một dòng cho một liên kết có thể của các kí hiệu mệnh đề trong câu. Mỗi dòng chúng ta có thể tính toán giá trị chân trị của toàn bộ câu. Nếu một câu là *true* ở mọi dòng, thì câu là hợp lệ. Ví dụ, cho câu sau:

$$((P \vee H) \wedge \neg H) \Rightarrow P$$

là hợp lệ, có thể xem trong Bảng 4-2. Bảng 4-2 có thêm các cột trung gian nhằm làm rõ hơn giá trị của cột cuối cùng. Giả sử mệnh đề *P* có

nghĩa "*Ngày mai trời mưa*" và *H* có nghĩa "*Ngày mai trời nắng*". Nếu chúng ta biết *(P ∨ H)* và *¬H*, thì chúng ta có thể sử dụng câu hợp lệ ở trên để kết luận *P* là đúng ngày mai trời mưa.

Bảng 4-2. *Bảng chân trị cho thấy tính hợp lệ của câu phức*

P	H	P ∨ H	(P ∨ H) ∧¬H	(P ∨ H) ∧¬H ⇒ P
False	False	False	False	True
False	True	True	False	True
True	False	True	True	True
True	True	True	False	True

Đây là một vấn đề quan trọng của logic. Một hệ thống có một vài tiền đề và các kết luận có khả năng xảy ra, hệ thống có thể xác định kết luận có đúng hay không. Có thể làm điều này bằng cách xây dựng một bảng chân trị cho câu *Tiền đề* ⇒ *Kết luận* và kiểm tra tất cả các dòng của nó. Nếu mọi dòng đều đúng, thì kết luận là suy ra được từ tiền đề. Mặc dù hệ thống chẳng biết kết luận nghĩa là gì, người dùng có thể đọc các kết luận và sử dụng cách hiểu các kí hiệu mệnh đề của mình để biết kết luận mang ý nghĩa gì.

Trong một số trường hợp thì các câu thường được do người dùng đưa vào cơ sở tri thức để mô tả về thế giới. Người dùng quan sát thế giới và nhập các câu vào máy tính. Yêu cầu của một hệ thống là có thể đưa ra những kết luận từ những tiền đề đã biết. Các câu có thể có ý nghĩa khác nhau nhưng tất cả đều có chung một nguyên lý hoạt động. Đó là hệ thống cần giải quyết bài toán suy dẫn logic.

II.4 Bài toán suy dẫn trong logic mệnh đề.

Bài toán suy dẫn rong logic được phát biểu như sau: cho một cơ sở tri thức ký hiệu KB bao gồm các câu được viết trong logic mệnh đề. Một câu logic α được gọi là suy dẫn được từ KB, ký hiệu KB ⊨ α khi và chỉ khi mọi cách hiểu làm cho KB đúng cũng làm cho α đúng, hay nói cách khác, mọi dòng trong bảng chân trị của câu KB ⊨ α đều đúng. Ví dụ một cơ sở tri thức biểu diễn cho các sự kiện sau:

- Hôm nay trời nắng: S.
- Nếu trời nắng thì Tuấn vui vẻ: S ⇒ H.

- Nếu Tuấn vui vẻ thì bài giảng tốt: H $\Rightarrow$ G.

Chứng minh rằng sự kiện Tuấn vui vẻ (G) được suy dẫn từ cơ sở tri thức trên.

KB = {S, S $\Rightarrow$ H, H $\Rightarrow$ G}.

α = G.

KB $\models \alpha$?.

Việc chứng minh cho suy dẫn này có thể được thực hiện dễ dàng thông qua bảng chân trị. Thật vậy, bảng chân trị cho bài toán này có ba mệnh đề đơn nên có 2^3=8 dòng. Trong 8 dòng đó, chỉ có một dòng duy nhất là cho cả ba câu trong KB đúng (*S = True*, *H = True* và *G = true*) trong đó *G* cũng là *True*. Nên KB là suy dẫn được α.

Cách suy dẫn dựa vào bảng chân trị là đúng và đầy đủ nhưng không hiệu quả. Để ý rằng số dòng trong bảng chân trị là 2^n với *n* là số mệnh đề đơn được phát biểu trong cơ sở tri thức. Bài toán tìm các dòng đúng trong bảng 2^n dòng rõ ràng là không khả thi khi *n* lớn. Do đó, những cách tiếp cận khác đã được đưa ra nhằm giải quyết vấn đề này. Các cách tiếp cận được xếp vào các hệ thống suy dẫn khác nhau dựa vào các quy tắc kiểm tra mà chúng thực hiện.

III. CÁC CÁCH SUY DIỄN TRONG LOGIC MỆNH ĐỀ

III.1 Hợp giải

Hệ thống hợp giải là một hệ thống chứng minh đúng và đủ đối với bài toán suy dẫn. Hệ thống này dựa trên một luật duy nhất được gọi là luật hợp giải được phát biểu như sau: cho các mệnh đề α, β, γ, khi đó:

$$\frac{\alpha \vee \neg\beta \qquad \beta \vee \neg\gamma}{\alpha \vee \gamma}$$

hay $(\alpha \vee \neg\beta) \wedge (\beta \vee \neg\gamma) \Rightarrow (\alpha \vee \gamma)$

Luật hợp giải trên là đúng. Ta dễ dàng kiểm chứng như: sau nếu α đúng thì rõ ràng kết luận là đúng và luật là đúng. Ngược lại nếu α sai thì β cũng phải sai (do câu $\alpha \vee \neg\beta$) và do đó γ là đúng vì $(\beta \vee \gamma)$ đúng.

Vậy suy ra kết luận đúng. Có thể thấy luật hợp giải là sự mở rộng tổng quát của luật tam đoạn luận (nếu $\alpha \Rightarrow \beta$ và α đúng thì β đúng). Một hệ thống chỉ cần sử dụng một mình luật hợp giải cũng là đủ để chứng minh cho một bài toán suy dẫn. Tuy nhiên, để áp dụng được phương pháp hợp giải, cơ sở tri thức cần được biến đổi để đưa về dạng chuẩn hội.

III.1.1 *Dạng hội chuẩn.*

Luật hợp giải chỉ áp dụng được lên các câu nối rời của các từ, do đó nó chỉ thích hợp với các cơ sở tri thức và truy vấn chứa các phép hội như trên. Như vậy làm thế nào phương pháp này có thể thực hiện một phép suy diễn đầy đủ trên tất cả logic mệnh đề? Câu trả lời là *mọi câu logic mệnh đề đều tương đương logic với một phép nối liền các phép hội của từ*. Một câu logic được biểu diễn là nối liền các phép hội của từ được gọi là **dạng hội chuẩn** (Conjunctive Normal Form) hay **CNF**. Vídụ: câu sau đây là dạng hội chuẩn:

$$A \wedge (B \vee \neg D) \wedge (B \vee E \vee F) \wedge \neg C.$$

Trong câu trên, mỗi từ là một ký hiệu mệnh đề hay phủ định của ký hiệu mệnh đề. Các phép hội được thực hiện trên các từ và cuối cùng được nối với nhau bằng phép nối liền.

Thay vì chứng minh nhận định rằng: Các câu logic mệnh đề đều tương đương logic với một câu dạng hội chuẩn, ta sẽ mô tả một thủ tục biến đổi đơn giản. Thủ tục biến đổi gồm các bước như sau:

1. Loại bỏ $\Leftrightarrow$, thay $\alpha \Leftrightarrow \beta$ bằng $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
2. Loại bỏ $\Rightarrow$, thay $\alpha \Rightarrow \beta$ bằng $\neg\alpha \vee \beta$
3. CNF đòi hỏi dấu phủ định $\neg$ chỉ được xuất hiện trên các từ, do đó cần phân phối dấu phủ định $\neg$ vào bên trong:

   $\neg(\neg\alpha) \equiv \alpha$

   $\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ (luật De Morgan)

   $\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ (luật De Morgan)

4. Bây giờ ta có các câu bao gồm các phép toán ∧ và ∨ lồng nhau trên các từ. Áp dụng luật phân phối ∨ vào ∧ cho tất cả trường hợp có thể:

$$\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$$

Ví dụ: với câu $A \Leftrightarrow (B \vee C)$, kết quả các bước biến đổi lần lượt là:

1. $(A \Rightarrow (B \vee C)) \wedge ((B \vee C) \Rightarrow A)$
2. $(\neg A \vee B \vee C) \wedge (\neg(B \vee C) \vee A)$
3. $(\neg A \vee B \vee C) \wedge ((\neg B \wedge \neg C) \vee A)$
4. $(\neg A \vee B \vee C) \wedge (\neg B \vee A) \wedge (\neg C \vee A)$

III.1.2 ***Thuật toán hợp giải,***

Thủ tục suy diễn dựa trên hợp giải hoạt động dựa trên nguyên tắc chứng minh phản chứng. Nghĩa là, để chứng minh câu $KB \models \alpha$ (KB suy dẫn hay kéo theo α), ta chứng minh phủ định của nó ($KB \wedge \neg\alpha$) không thoả. Việc này được chứng minh bằng cách chỉ ra một mâu thuẫn bên trong các mệnh đề.

Giải thuật hợp giải được trình bày trong Hình 4.1. Đầu tiên ($KB \wedge \neg\alpha$) được biến đổi về dạng CNF. Sau đó, luật hợp giải được áp dụng lên các mệnh đề kết quả. Mỗi cặp mệnh đề chứa các từ bù nhau được hợp giải để tạo ra mệnh đề mới và được thêm vào tập câu nếu nó chưa xuất hiện. Quá trình tiếp tục đến khi một trong hai điều sau xảy ra:

- Không có mệnh đề mới nào có thể được thêm vào, trong trường hợp đó KB không suy dẫn được α
- Hai mệnh đề hợp giải nhận được một mệnh đề ***rỗng***, KB suy dẫn được α

Xét ví dụ sau cho chứng minh hợp giải:

$KB = A \Leftrightarrow (B \vee C)$

$\alpha = (A \Rightarrow B)$

Lấy phủ định mệnh đề α và đưa vào tập hợp ta được:

$KB = (A \Leftrightarrow (B \vee C)) \wedge \neg(A \Rightarrow B)$

Biến đổi về CNF:

```
function HợpGiải(KB,α) return true hay false
input:KB, cơ sở tri thức, câu logic mệnh đề
      α,câu truy vấn, câu logic mệnh đề
mệnh_đề← tập các câu biểu diễn CNF của KB ∧¬α
mđ_mới← {}
loop
      for each C_i, C_j in tập mệnh_đề
            kết_quả_hợp_giải←HợpGiảiCâu(C_i, C_j)
            nếu kết_quả_hợp_giải chứa mệnh đề rỗng thì trả về true
            mđ_mới←mđ_mới Ukết_quả_hợp_giải
      nếu mđ_mới⊆mệnh_đề thì trả về false
      mệnh_đề←mệnh_đề Umđ_mới
```

Hình 4.1 - *Thuật toán hợp giải*

$KB = (\neg A \vee B \vee C) \wedge (\neg B \vee A) \wedge (\neg C \vee A) \wedge \neg A \wedge B$

với các câu hội:

1. $\neg A \vee B \vee C$
2. $\neg B \vee A$
3. $\neg C \vee A$
4. $\neg A$
5. B

Hợp giải các câu theo thức tự sau:

6.	$\neg A \vee A \vee C$	(1,2)
7.	$\neg B \vee B \vee C$	(1,2)
8.	$\neg A \vee A \vee B$	(1,3)
9.	$\neg C \vee B \vee C$	(1,3)
10.	$\neg B$	(2,4)
11.	$\neg C$	(3,4)
12.	{}	(5,10) (câu rỗng, kết thúc chứng minh)

Có một số lưu ý trong ví dụ chứng minh trên:

- Một cặp mệnh đề hợp giải với nhau có thể ra nhiều kết quả, như câu 1 và 2 ra 6 và 7, hoặc 1 và 3 ra 8 và 9, khi có nhiều cặp từ bù giữa hai câu. Tuy nhiên, các câu kết quả ***không*** đóng

góp vào quá trình chứng minh đúng (do các câu trên luôn còn lại mệnh đề mỗi khi hợp giải) do đó ta có thể loại bỏ.

- Việc chứng minh kết thúc hết cứ khi nào có mệnh đề rỗng (câu 12) nên tuỳ thuộc vào thứ tự các bước thực hiện, việc chứng minh có thể kết thúc sớm hay muộn. Một hệ thống đòi hỏi vét cạn sẽ duyệt qua hết các trường hợp hợp giải. Tuy nhiên, chứng minh của ví dụ trên có thể được thực hiện một cách nhanh chóng bằng cách hợp giải các câu (2) và (4) rồi đến câu (5), chỉ qua hai bước hợp giải.

III.2 Suy diễn tiến

Tính đầy đủ của hợp giải khiến nó trở thành một phương pháp suy diễn rất quan trọng. Tuy nhiên, trong nhiều tình huống sức mạnh đầy đủ của hợp giải là không cần thiết. Các cơ sở tri thức trong thế giới thực thường chỉ chứa các mệnh đề ở dạng hạn chế gọi là các mệnh đề Horn. Một mệnh đề Horn là một nối rời các từ mà trong đó chỉ có *nhiều nhất* một từ khẳng định. Ví dụ, mệnh đề ($\neg A \vee \neg B \vee C$) là một mệnh đề Horn và ($\neg A \vee B \vee C$) không phải là mệnh đề Horn.

Điểm hạn chế chỉ có một từ khẳng định có vẻ hơi bất thường và không thú vị, nhưng đây là điểm thật sự rất quan trọng vì ba lý do sau:

1. Mỗi mệnh đề Horn có thể viết về dạng kéo theo mà trong đó tiền đề là một phép nối rời giữa các từ khẳng định và kết luận là một từ khẳng định duy nhất. Ví dụ, mệnh đề Horn ($\neg A \vee \neg B \vee C$) có thể viết lại thành dạng kéo theo ($A \wedge B \Rightarrow C$). Câu ở dạng thứ hai dễ đọc và hiểu hơn đối với con người: nếu A đúng và B đúng thì C cũng đúng. Mệnh đề Horn dạng này với *đúng* một từ khẳng định được gọi là mệnh đề xác định. Từ khẳng định được gọi là phần đầu và các từ phủ định hình thành nên phần thân của mệnh đề. Một mệnh đề xác định không có từ phủ định đơn giản được xem là một mệnh đề cho trước - đôi khi được gọi là sự kiện. Một mệnh đề Horn không có từ khẳng định có thể viết về dạng kéo theo với kết luận là từ *False*.

2. Suy diễn với mệnh đề Horn có thể được thực hiện qua các thuật toán **suy diễn tiến** và **suy diễn lùi**. Cả hai thuật toán này

đều rất tự nhiên, theo nghĩa các bước suy diễn là rõ ràng và dễ hiểu đối với con người.

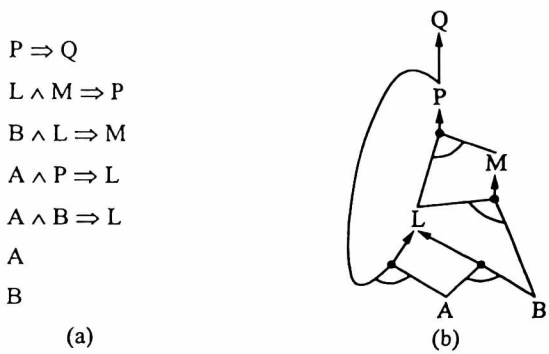
3. Việc chứng minh bài toán suy dẫn với mệnh đề Horn có thể thực hiện trong thời gian tuyến tính với độ lớn của cơ sở tri thức.

```
function Suy_diễn_tiến(KB,q) return true hay false
  input: KB, cơ sở tri thức, câu logic mệnh đề
         q.câu truy vấn, câu logic mệnh đề
  biến cục bộ:count, một bảngkhởi tạo số của các tiền đề, đánh chỉ mục
                     bởi các mệnh đề
              inferred, một bảng được đánh chỉ mục bằng các ký hiệu, mỗi
                     mục được khởi tạo false
              agenda, danh sách các ký hiệu, khởi tạo là các ký hiệu đã
                     biết của KB
  while agenda khác rỗng do
      p ← POP(agenda)
      if p=q then return true
      if inferred[p] là falsethen
      for each mệnh đề Horn c mà tiền đề xuất hiện p
              giảm count[c]
              if count[c] = 0 then
                     PUSH(Phần_đầu[c], agenda)
  return false
```

Hình 4.2 - *Thuật toán suy diễn tiến.*

Thuật toán suy diễn tiến Suy_diễn_tiến(KB,q) kiểm tra xem một ký hiệu truy vấn mệnh đề q - câu truy vấn - có được suy dẫn từ một cơ sở tri thức mệnh đề Horn hay không. Nó bắt đầu từ các sự kiện đã biết (từ khẳng định) trong cơ sở tri thức. Nếu tất cả tiền đề của một phép kéo theo là đã biết thì kết luận của nó được thêm vào tập các sự kiện đã biết. Ví dụ, nếu A và B đã biết và $(A \wedge B) \Rightarrow M$ nằm trong cơ sở tri thức thì M được thêm vào. Quá trình này tiếp tục cho đến khi câu truy vấn q được thêm vào hoặc không còn suy diễn nào có thể được thực hiện tiếp. Chi tiết thuật toán trình bày trong Hình 4.2, điểm quan trọng cần nhớ là thuật toán chạy với thời gian tuyến tính.

Cách tốt nhất để hiểu thuật toán là xét một ví dụ. Hình 4.3(a) trình bày một cơ sở tri thức đơn giản của mệnh đề Horn với A và B là các sự kiện đã biết. Hình 4.3(b) trình bày cùng cơ sở tri thức đó bằng một đồ thị AND-OR. Trong đồ thị AND-OR, nhiều liên kết đến một đỉnh được nối với nhau bằng cung tròn là một phép nối liền - mọi liên kết đều phải được chứng minh, trong khi những liên kết đến một đỉnh không được nối với nhau bằng cung tròn là một phép nối rời - có thể chứng minh bất kỳ liên kết nào. Dễ dàng nhìn thấy cách suy diễn tiến hoạt động trên đồ thị này. Những nút là đã biết (A và B) được kích hoạt và suy diễn lan truyền theo hướng từ dưới lên trên trong đồ thị càng xa càng tốt. Tại những điểm có phép nối rời, suy diễn chờ đến khi tất cả mệnh đề đều biết trước khi tiến hành lan truyền.



Hình 4.3 - *(a) Một cơ sở tri thức đơn giản các mệnh đề Horn.*
(b) Đồ thị AND-OR tương ứng.

Suy diễn tiến là một ví dụ của khái niệm suy diễn hướng dữ liệu–suy diễn tập trung bắt đầu tại dữ liệu đã biết. Nó có thể dùng trong một hệ thống để suy dẫn ra kết luận từ những cảm nhận đầu vào. Đối với con người, một số lượng nhất định suy diễn hướng dữ liệu diễn ra khi tiếp nhận thông tin mới. Ví dụ, nếu tôi đang ở trong nhà và thấy trời bắt đầu mưa, một điều sẽ nảy sinh ra trong đầu là có thể buổi đi chơi sẽ bị huỷ bỏ. Nhưng có thể tôi sẽ không quan tâm rằng quần áo của người

hàng xóm bên cạnh sẽ bị ướt. Con người kiểm soát rất cẩn thận suy diễn tiến để trí óc ta không bị chìm ngập trong những kết luận không liên quan.

III.3 Suy diễn lùi

Suy diễn lùi, như tên gọi của nó, hoạt động quay lui từ câu truy vấn. Nếu truy vấn q đã được biết là đúng thì không cần làm gì thêm. Ngược lại, thuật toán tìm những phép suy dẫn trong cơ sở tri thức có kết luận là q. Nếu tất cả tiền đề của một trong số những phép suy dẫn có thể được chứng minh là đúng (bằng suy diễn lùi), thì q là đúng. Khi được áp dụng lên truy vấn Q trong Hình 4.3, thuật toán đi từ trên xuống dưới trên đồ thị cho đến khi gặp đến một tập các sự kiện đã biết hình thành nên cơ sở của việc chứng minh. Cũng như với suy diễn tiến, cài đặt hiệu quả của suy diễn lùi hoạt động với thời gian tuyến tính.

Suy diễn lùi là một dạng của suy diễn hướng đích. Nó hữu ích để trả lời các câu hỏi cụ thể như "Tôi cần làm gì bây giờ?" và "Thẻ xe của tôi đang ở đâu?" Thông thường chi phí của suy diễn lùi ít hơn nhiều so với tuyến tính độ lớn của cơ sở tri thức, bởi vì quá trình suy diễn chỉ dùng đến những sự kiện có liên quan. Nói chung, một hệ thống có thể chia sẻ công việc giữa suy diễn tiến và lùi, giới hạn suy diễn tiến trên những thế hệ sự kiện có thể có liên quan đến truy vấn sẽ được giải quyết bằng suy diễn lùi.

IV. KẾT CHƯƠNG

Chương này giới thiệu các hệ thống dựa trên tri thức và trình bày cách định nghĩa logic giúp hệ thống có thể thực hiện các suy diễn trên cơ sở tri thức đó. Một số điểm chính của hệ thống logic như sau:

- Tri thức được chứa trong các hệ thống dưới dạng các câu trong một ngôn ngữ biểu diễn tri thức, được lưu trữ trong một cơ sở tri thức.
- Các hệ thống dựa trên tri thức được tạo thành bởi một cơ sở tri thức và một cơ chế suy diễn. Hệ thống hoạt động bằng cách lưu trữ các câu mô tả thế giới trong cơ sở tri thức, sử dụng cơ

chế suy diễn để rút ra các câu mới và sử dụng những câu này để quyết định hành động.

- Một ngôn ngữ biểu diễn được định nghĩa bởi cú pháp và ngữ nghĩa. Cú pháp xác định cấu trúc của các câu trong ngôn ngữ và ngữ nghĩa định nghĩa chân trị của từng cấu trong mỗi mô hình.
- Mối quan hệ kéo theo giữa các câu là quan trọng nhất đối với bài toán suy dẫn. Một câu *a* kéo theo một câu *b* khác nếu *b* đúng trong tất cả các mô hình mà *a* đúng.
- Suy diễn là quá trình rút ra các câu mới từ câu cũ. Thuật toán suy diễn là đúng đắn nếu nó chỉ rút ra các câu được kéo theo và thuật toán là đầy đủ nếu nó rút ra tất cả các câu được kéo theo.
- Logic mệnh đề là một ngôn ngữ rất đơn giản bao gồm các ký hiệu mệnh đề và các phép liên kết logic.
- Các luật suy diễn là các phương thức suy diễn đúng đắn có thể dùng để tìm các chứng minh. Luật hợp giải cung cấp một thuật toán suy diễn đầy đủ đối với các cơ sở tri thức biểu diễn dưới dạng hội chuẩn. Suy diễn tiến và suy diễn lùi là những thuật toán suy diễn rất tự nhiên cho các cơ sở tri thức ở dạng Horn.
- Logic mệnh đề có hiệu quả lý tưởng đối với một số nhiệm vụ nhất định trong một hệ thống, nhưng không đủ linh hoạt trong các môi trường thế giới thực, vì nó thiếu khả năng biểu diễn để xử lý chính xác thời gian, không gian và các mẫu quan hệ giữa các đối tượng.

V. THUẬT NGỮ TIẾNG ANH

Sau đây là những thuật ngữ tiếng Anh liên quan đến những khái niệm đã dùng ở chương này:

atomic sentence: câu đơn

backward reasoning/chaining: suy diễn lùi

biconditional: tương đương

clause: mệnh đề

complex sentence: câu phức

conjunction: phép logic và

Conjunctive Normal Form (CNF): dạng hội chuẩn

data-driven inference: suy diễn hướng dữ liệu

disjunction: phép logic hoặc

entailment: phép kéo theo

equivalance: tính tương đương

forward reasoning/chaining: suy diễn lùi

goal-driven reasoning: suy diễn hướng đích

Horn clause: mệnh đề Horn

implication: kéo theo

inference: suy dẫn

literal: từ (logic)

logical connective: liên kết logic

negation: phủ định

proof by contraction: chứng minh phản chứng

proof theory: lý thuyết chứng minh

propositional logic: logic mệnh đề

propositional symbol: ký hiệu mệnh đề

reasoning: suy diễn

resolution: hợp giải

satisfiability: tính thoả mãn được

semantics: ngữ nghĩa

syntax: cú pháp

truth table: bảng chân trị

validity: tính hợp lệ

Chương 5
LOGIC BẬC NHẤT

Chương 4 đã đề cập đến cách biểu diễn một thế giới và tri thức suy luận được từ biểu diễn đó. Phương pháp được sử dụng là logic mệnh đề, ngôn ngữ thể hiện này minh họa các khái niệm cơ bản của logic và các tác nhân cơ cở tri thức. Tuy nhiên, logic mệnh đề quá yếu để thể hiện tri thức của môi trường phức tạp theo một cách đơn giản hơn. Trong chương này, chúng ta sẽ cùng tìm hiểu về logic bậc nhất (first-order logic) và cách giải quyết vấn đề một cách hiệu quả.

I. CÚ PHÁP VÀ NGỮ NGHĨA CỦA LOGIC BẬC NHẤT

I.1 Mô hình của logic bậc nhất

Trong chương trước, chúng ta có thể thấy mô hình của logic mệnh đề chỉ là tập hợp của các giá trị chân trị của những biến mệnh đề. Ngược lại mô hình của logic bậc nhất có nhiều điểm nổi trội hơn. Đầu tiên, mô hình này chứa các đối tượng. Miền xác định của mô hình là tập hợp của những đối tượng chứa trong đó, những đối tượng này được gọi là các phần tử miền xác định. Hình 5.1 là một mô hình gồm 5 đối tượng: Hoàng, vua của một nước; em trai của anh ấy,VuaVương, người thừa kế ngai vàng sau đó; chân trái của Hoàng và Vương; cuối cùng là cái vương miện.

Các đối tượng trong mô hình liên hệ với nhau theo nhiều cách khác nhau như Hoàng và Vương là anh em trai. Trong ngôn ngữ nói, một quan hệ chính là một bộ ba những đối tượng có liên quan với nhau. Mối quan hệ "anh em" trong mô hình này là tập hợp:

{(Vua Hoàng, Vua Vương), (Vua Vương, Vua Hoàng)}

Vương miện được đặt trên đầu của Vua Vương, vì vậy quan hệ "trên đầu" chỉ chứa một bộ, {(vương miện, Vua Vương)}. Mối quan hệ "anh em" và "trên đầu" được gọi là mối quan hệ hai ngôi do nó có một cặp đối tượng. Mô hình có thể chứa quan hệ một ngôi, được gọi là quan hệ thuộc tính: thuộc tính "người" đúng cho cả Hoàng và Vương; thuộc tính "vương miện" chỉ đúng cho chiếc vương miện.

Quan hệ được xem như một hàm khi những đối tượng cho trước liên quan đến một đối tượng khác. Ví dụ, mỗi người có một chân trái, do đó, chân trái của Hoàng phân biệt với chân trái của Vương. Một cách chặt chẽ, trong mô hình logic bậc nhất, hàm chỉ trả về duy nhất một giá trị từ các giá trị đầu vào. Giải pháp này gặp phải vấn đề là nó tính cả những trường hợp không thể xảy ra, chẳng hạn chúng ta không thể tìm thấy chân trái của những thứ không có chân trái. Tuy nhiên, một điều may mắn là chúng ta không phải truy vấn những câu không thể xảy ra như vậy.

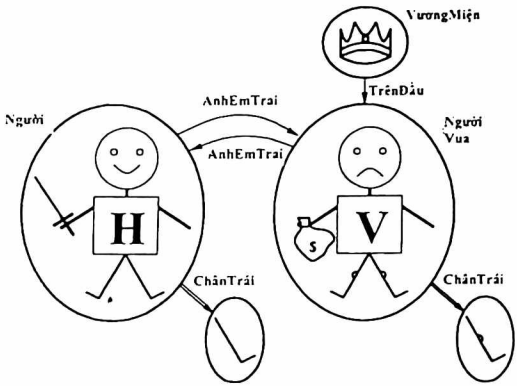
Vương miện được đặt trên đầu của Vua Vương, vì vậy quan hệ "trên đầu" chỉ chứa một bộ, {(vương miện, Vua Vương)}. Mối quan hệ "anh em" và "trên đầu" được gọi là mối quan hệ hai ngôi do nó có một cặp đối tượng. Mô hình có thể chứa quan hệ một ngôi, được gọi là quan hệ thuộc tính: thuộc tính "người" đúng cho cả Hoàng và Vương; thuộc tính "vương miện" chỉ đúng cho chiếc vương miện.

Quan hệ được xem như một hàm khi những đối tượng cho trước liên quan đến một đối tượng khác. Ví dụ, mỗi người có một chân trái, do đó, chân trái của Hoàng phân biệt với chân trái của Vương. Một cách chặt chẽ, trong mô hình logic bậc nhất, hàm chỉ trả về duy nhất một giá trị từ các giá trị đầu vào. Giải pháp này gặp phải vấn đề là nó tính cả những trường hợp không thể xảy ra, chẳng hạn chúng ta không thể tìm thấy chân trái của những thứ không có chân trái. Tuy nhiên, một điều may mắn là chúng ta không phải truy vấn những câu không thể xảy ra như vậy.

I.2 Ký hiệu và biểu diễn

Thành phần cú pháp cơ bản nhất của logic bậc nhất là các kí hiệu đại diện cho các đối tượng, mối quan hệ và hàm. Do đó có thể chia làm ba loại kí hiệu chính: kí hiệu hằng đại diện cho đối tượng; kí hiệu vị từ

đại diện cho quan hệ và kí hiệu hàm đại hiện cho hàm. Ta sẽ qui ước các kí hiệu này bắt đầu bằng kí tự viết hoa. Ví dụ chúng ta có các kí hiệu hằng là Hoàng, Vương; kí hiệu vị từ như AnhEmTrai, TrênĐầu, VịVua và VươngMiện; kí hiệu hàm như ChânTrái.



Hình 5.1 - *Một mô hình chứa năm đối tượng, hai quan hệ hai ngôi, ba quan hệ một ngôi (được xác định bởi nhãn của đối tượng) và một hàm một ngôi, chân trái*

Ngữ nghĩa phải liên hệ câu phát biểu với mô hình để xác định chân trị. Điều này có thể thực hiện được khi chúng ta sử dụng các suy diễn, từ đó biết được đối tượng, quan hệ hay hàm nào được chỉ đến bởi các kí hiệu hằng, kí hiệu vị từ và kí hiệu hàm.

Mô hình có thể có số lượng các đối tượng vô hạn như mô hình số tự nhiên, số thực,... do đó các suy diễn cũng không có giới hạn.

Tóm lại, các hằng số thể hiện những đối tượng thực sự của thế giới, các biến thể hiện những đối tượng chưa biết.

Hàm thể hiện đối tượng dựa trên các đối tượng khác. Hàm được xem như là một dạng biểu thức.

Câu → *Câu Nguyên Tố*

| *Câu nối Câu sử dụng phép kết*

| *Biến Lượng Từ, ...Câu*

| ¬ *Câu*

Câu Nguyên Tố → *Vị Từ (Biểu Thức, ...)* | *Biểu Thức* = *Biểu Thức*

Biểu Thức → *Hàm(Biểu Thức, ...)*

| *Hằng*

| *Biến*

Phép Kết → ⇒ | ∧ |∨ | ⇔

Lượng từ → ∀ | ∃

Hằng → *A* | X_1 | *Vương* | ...

Biến → *a* | *x* | *s* | ...

Vị Từ → *Trước* | *CóMàu* | ...

Hàm → *MẹCủa* | *ChânTráiCủa* | ...

Hình 5.2 - *Cú pháp của logic bậc nhất với toán tử tương đương*

Vị từ là một hàm đặc biệt (nhưng không phải là một biểu thức) thể hiện các mối quan hệ với hai trạng thái {true, false}, nếu tất cả các đối số của vị từ được gán giá trị xác định hay không còn biến xuất hiện ở đối số thì nó trở thành một mệnh đề vì lúc này chân trị đã được xác định. Một vị từ có duy nhất một biến thường được gọi là thuộc tính. Ví dụ: *ConMèo(x)* là true nếu và chỉ nếu *x* là một con mèo.

I.3 Biểu thức

Một biểu thức (term) được sử dụng để chỉ đến các đối tượng trong miền đang xét. Vì vậy, kí hiệu hằng số chính là một biểu thức logic. Tuy nhiên, không phải lúc nào mỗi đối tượng cũng được đặt tên bởi một kí hiệu riêng biệt, ví dụ, "chân trái của Vương" sẽ chỉ đến tên của chân anh, thay vì phải sử dụng thêm một kí hiệu hằng số nữa, chúng ta

sử dụng kí hiệu hàm ChânTrái(Vương). Nhìn chung, một biểu thức phức tạp được sinh ra từ một kí hiệu hàm trong đó một danh sách các biểu thức logic đóng vai trò như tham số đầu vào.

I.4 Câu nguyên tố

Câu nguyên tố được hình thành từ một kí hiệu vị từ theo sau bởi danh sách đối số là các biểu thức logic. Câu nguyên tố sẽ chỉ đến một sự kiện. Ví dụ:

AnhEmTrai (Hoàng, Vương).

Cây này chỉ ra sự kiện, Hoàng là anh em trai của Vương. Câu nguyên tố có thể chứa các biểu thức phức tạp làm đối số.

KếtHôn(Cha(Hoàng),Mẹ(Vương)).

Có nghĩa là cha của Hoàng kết hôn với mẹ của Vương.

Một câu nguyên tố chỉ đúng trong mô hình và các suy diễn cho trước, khi và chỉ khi mối quan hệ được biểu diễn bởi vị từ thực sự xảy ra giữa các đối tượng làm đối số của vị từ.

Câu nguyên tố cơ sở là câu mà tất cả các biểu thức làm đối số đều không chứa biến, như vậy cũng có nghĩa là chân trị của nó đã được xác định (mệnh đề).

I.5 Câu phức

Chúng ta có thể sử dụng các phép kết logic để xây dựng những câu phức giống như tính toán mệnh đề. Ngữ nghĩa của câu được hình thành với các phép liên kết logic tương tự như trong trường hợp mệnh đề. Ví dụ:

$\neg AnhEmTrai(ChânTrái(Hoàng), Vương).$

$AnhEmTrai(Hoàng, Vương) \wedge AnhEmTrai(Vương, Hoàng).$

$Vua(Hoàng) \vee Vua(Vương).$

$\neg Vua(Hoàng) \Rightarrow Vua(Vương).$

I.6 Lượng từ

Đôi khi, ta muốn nhấn mạnh thuộc tính không phải chỉ là một số đối tượng đã được định danh, mà là toàn bộ các đối tượng. Lượng từ sẽ giúp chúng ta điều này. Logic bậc nhất chứa hai lượng từ, được gọi là lượng từ phổ quát hay "với mọi" và lượng từ "tồn tại".

Một câu viết đúng công thức là một câu chứa các biến bị giới hạn bởi các lượng từ *phổ quát* (hay lượng từ với mọi) và *tồn tại*. Ví dụ: Đối với câu $\forall x\ P(x, y)$ thì biến x được giới hạn bởi lượng từ phổ quát nhưng biến y thì tự do.

I.6.1 Lượng từ phổ quát

Cho câu "Tất cả các vị vua đều là người", được viết trong logic bậc nhất như sau:

$$\forall x\ Vua(x) \Rightarrow Người(x).$$

Kí hiệu: $\forall$ có nghĩa là "với mọi...", do đó câu này được phát biểu "với mọi x, nếu x là vua, thì x là người". Kí hiệu x gọi là biến và được viết thường. Một biến có thể là một biểu thức và như vậy có thể đóng vai trò như đối số của hàm. Một biểu thức không có biến gọi là biểu thức cơ sở.

Câu $\forall x$ P, trong đó P là biểu thức logic bất kì, nghĩa là P đúng cho mọi đối tượng x. Chặt chẽ hơn ta nói $\forall x$ P chỉ đúng trong mô hình với những suy diễn cho trước nếu P đúng trong mọi suy diễn mở rộng hình thành từ những suy diễn có trước đó, trong đó mỗi suy diễn mở rộng xác định một phần tử thuộc mô hình mà x chỉ đến. Xem xét ví dụ sau để hiểu hơn về phát biểu này. Quay lại mô hình 5.1, tiến hành mở rộng suy diễn theo 5 cách:

$x \rightarrow$ Hoàng,

$x \rightarrow$ VuaVương,

$x \rightarrow$ chân trái của Hoàng,

$x \rightarrow$ chân trái của Vương,

$x \rightarrow$ vương miện.

Câu $\forall x$ *Vua(x)* $\Rightarrow$ *Người(x)* chỉ đúng trong suy diễn gốc nếu câu *Vua(x)* $\Rightarrow$ *Người(x)* đúng với mỗi suy diễn mở rộng trên.

Hoàng là một vị vua $\Rightarrow$ Hoàng là người.

VuaVương là một vị vua $\Rightarrow$ VuaVương là người.

Chân trái của Hoàng là một vị vua $\Rightarrow$ Chân trái của Hoàng là người.

Chân trái của Vương là một vị vua $\Rightarrow$ Chân trái của Vương là người.

Vương miện là một vị vua $\Rightarrow$Vương miện là người.

Trong mô hình 5.1, chỉ có VuaVương là vị vua, như vậy câu thứ hai kết luận anh ấy là người là điều chúng ta mong muốn. Nhưng 4 câu còn lại thì sao? Nó có phải là một phần ý nghĩa của câu "mọi vị vua là người" không? Thực ra, 4 câu này là đúng nhưng nó không cho biết liệu cái chân, vương miện, Hoàng có thuộc nhóm người hay không bởi các đối tượng này không phải là một vị vua. Chúng ta sẽ thấy dễ dàng qua bảng chân trị của phép kéo theo ($\Rightarrow$), suy luận vẫn đúng trong trường hợp điều kiện bị sai (false) mà không cần quan tâm kết luận như thế nào.

Một lỗi thường gặp trong khi sử dụng lượng từ phổ quát, đó là sử dụng phép nối liền thay vì phép kéo theo. Ví dụ:

$$\forall x \; Vua(x) \wedge Người(x).$$

sẽ có nghĩa là:

Hoàng là một vị vua $\wedge$ Hoàng là người.

Vương là một vị vua $\wedge$ Vương là người.

Chân trái của Hoàng là một vị vua $\wedge$ Chân trái của Hoàng là người.

Chân trái của Vương là một vị vua $\wedge$ Chân trái của Vương là người.

Vương miện là một vị vua $\wedge$ Vương miện là người.

Dựa vào bảng chân trị của phép ∧ (và) có thể thấy câu trên không thể đúng với mọi x.

Ví dụ khác:

$\forall x$ *Ở(x, KHTN)* $\Rightarrow$ *ThôngMinh(x).*

Mọi người ở KHTN thì thông minh.

$\forall x$ *Ở(x, KHTN)* $\wedge$ *ThôngMinh(x).*

Mọi người ở KHTN và mọi người thông minh.

I.6.2 Lượng từ tồn tại

Ví dụ, "VuaVương có một vương miện trên đầu ông ta", được viết:

$\exists x$ *VươngMiện(x)*$\wedge$ *TrênĐầu(x, Vương).*

Kí hiệu: $\exists$ có nghĩa là "tồn tại một x sao cho..." hay "một số x ...".

$\exists x$ P nghĩa là P đúng khi ít nhất một x thỏa. Chặt chẽ hơn ta nói $\exists x$ P chỉ đúng trong mô hình, với những suy diễn cho trước nếu P đúng trong *ít nhất một* suy diễn mở rộng mà nó đã gán x đến một phần tử xác định của miền. Quay lại ví dụ trên, nghĩa của câu đúng khi ít nhất một trong các câu sau phải đúng:

Hoàng là một vương miện ∧ Hoàng nằm trên đầu của Vương;

Vua Đế là một vương miện ∧ Vua Đế nằm trên đầu của Vương;

Chân của Hoàng là một vương miện ∧ chân của Hoàng nằm trên đầu của Vương;

Chân của Đế là một vương miện ∧ chân của Đế nằm trên đầu của Vương;

Vương miện là một vương miện ∧ vương miện nằm trên đầu của Vương.

Câu cuối cùng đúng trong mô hình, vì vậy câu có lượng từ tồn tại đúng trong mô hình này.

Thường người ta sử dụng phép "và" (∧) đi cùng với lượng từ tồn tại. Nếu sử dụng ⇒ đi cùng với lượng từ tồn tại sẽ dẫn đến câu không đúng, chẳng hạn:

$$\exists x\ VươngMiện(x) \Rightarrow TrênĐầu(x, Vương).$$

Chỉ cần một trong số câu theo sau đúng, sẽ làm câu trên đúng:

Hoàng là một vương miện ⇒ Hoàng nằm trên đầu của Vương;

VuaĐế là một vương miện ⇒ VuaĐế nằm trên đầu của Vương;

...

Theo tính chất của phép kéo theo, câu giả thiết sai sẽ dẫn đến việc suy diễn là đúng. Chính vì vậy, mới vào câu đầu, chúng ta đã thỏa mãn phép ∃ mà không cần xét thêm. Điều này làm cho câu chưa chắc đúng về mặt ngữ nghĩa.

Ví dụ khác:

$\exists x\ Ở(x, KHTN) \wedge ThôngMinh(x).$

Một số người ở KHTN thì thông minh.

$\exists x\ Ở(x, KHTN) \Rightarrow ThôngMinh(x).$

Câu này đúng cả với trường hợp một ai đó không ở KHTN.

.1.6.3 Lượng từ lồng nhau

Có thể sử dụng nhiều lượng từ lồng nhau để minh họa các câu phức tạp hơn. Trường hợp đơn giản nhất là các lượng từ cùng loại. Ví dụ, nói "anh em trai thì có cùng cha mẹ" được viết như sau:

$$\forall x\ \forall y\ AnhEmTrai(x,y) \Rightarrow CùngChaMẹ(x,y).$$

Chuyển đổi thứ tự của lượng từ cùng loại sẽ không làm thay đổi nghĩa của câu:

$$\forall x\ \forall y\ P(x, y) \Leftrightarrow \forall y \forall x\ P(x, y)$$

$$\exists x\ \exists y\ P(x, y) \Leftrightarrow \exists y \exists x\ P(x, y)$$

Để súc tích hơn, đối với các câu có cùng lượng từ, người ta có thể viết một kí hiệu lượng từ duy nhất, theo sau là các biến:

$$\forall\ x, y\ AnhEmTrai(x,y) \Rightarrow CùngChaMẹ(x,y).$$

Trong trường hợp có sự trộn lẫn của hai loại lượng từ, như "Mọi người đều yêu một ai đó", viết theo logic bậc nhất:

$$\forall x \exists y\ Yêu(x,y).$$

Ngược lại để nói "Có một ai đó yêu tất cả mọi người":

$$\exists y\ \forall x\ Yêu(y,x).$$

Do đó thứ tự của lượng từ rất quan trọng trong trường hợp có tồn tại cả hai loại lượng từ. Một nhập nhằng xảy ra nếu hai loại lượng từ này sử dụng chung một biến:

$$\forall x\ [VươngMiện(x) \vee (\exists x\ AnhEmTrai(Hoàng, x))]$$

Luật sẽ được áp dụng để phân xử trong trường hợp này, x sẽ mang ý nghĩa của lượng từ gần nó nhất về phía trái. Việc viết như vậy sẽ gây rối rắm cho các bước tìm lỗi sau này. Tốt nhất, nếu các biến mang ý nghĩa khác nhau phải đặt tên khác nhau.

I.6.4 Mối tương quan giữa $\forall$ và $\exists$

Hai lượng từ này có mối tương quan với nhau thông qua phép phủ định. Phát biểu "mọi người không thích ăn cải" cũng tương tự như "không có một ai mà người đó lại thích ăn cải cả":

$$\forall x\ \neg\ ThíchĂn(x,Cải) \text{ tương đương } \neg\ \exists x ThíchĂn(x,Cải).$$

Bằng cách áp dụng định luật đối ngẫu De Morgan, ta có thể liệt kê các mối tương quan giữa hai lượng từ:

$$\forall x \neg P \equiv \neg \exists x\ P \qquad \neg \forall x\ P \equiv \exists x\ \neg P \qquad \forall x\ P \equiv \neg\ \exists x\ \neg P$$

$$\exists x\ P \equiv \neg \forall x\ \neg P$$

I.7 Phép bằng

Logic bậc nhất có thêm một cách nữa để tạo ra những câu nguyên tố bên cạnh sử dụng vị từ và biểu thức như mô tả phần trước. Đó là sử dụng kí hiệu "bằng" để chỉ ra hai biểu thức cùng nói về một đối tượng. Ví dụ:

$$ChaCủa(Vương) = Thái$$

nghĩa là đối tượng mà hàm *ChaCủa(Vương)* chỉ đến và đối tượng *Thái* chỉ đến là giống nhau. Xác định chân trị của câu bằng nhau đơn giản là kiểm tra xem đối tượng của hai biểu thức có cùng tham chiếu đến có giống nhau hay không.

Chúng ta có thể sử dụng phép phủ định để chỉ ra hai biểu thức không cùng tham chiếu đến một đối tượng. Ví dụ, để nói Hoàng có ít nhất hai anh trai, chúng ta viết:

$$\exists x,y\ AnhEmTrai(x,Hoàng) \wedge AnhEmTrai(y, Hoàng) \wedge \neg(x=y).$$

Có thể viết tắt: $\neg(x=y)$ bằng $x \neq y$.

II. SỬ DỤNG LOGIC BẬC NHẤT

II.1 Ví dụ miền quan hệ "họ hàng"

Chúng ta sẽ xem xét các mối quan hệ họ hàng bao gồm các sự kiện như "Thái là cha của Vương"; "Mẫu là mẹ của Hoàng";... và các luật như "Bà ngoại là mẹ của mẹ".

Rõ ràng những đối tượng trong miền xác định này là người. Chúng ta có hai vị từ một ngôi về giới tính *GTNam* và *GTNữ*. Các mối quan hệ gia đình bao gồm cha mẹ, anh em trai, đã kết hôn,... – sẽ được thể hiện bởi các vị từ hai ngôi: *ChaMẹ, AnhChịEm, AnhEmTrai, ChịEmGái, ConCái, ConGái, ConTrai, VợChồng, Vợ, Chồng, ÔngBà,*

Cháu, AnhEmHọ, Cô, Chú. Các hàm là *Mẹ* và *Cha* vì mỗi người chỉ có một mẹ và một cha (theo tự nhiên).

Chúng ta tiến hành xem xét các vị từ và các hàm, viết ra những gì mà chúng ta biết sử dụng các biểu thức:

Mẹ là một trong hai người có quan hệ mẹ cha và có giới tính là nữ:

$$\forall m,c\ Mẹ(c) = m \Leftrightarrow GTNữ(m) \wedge ChaMẹ(m,c).$$

Chồng là một trong hai người có quan hệ vợ chồng với nhau và có giới tính là nam:

$$\forall w,h\ Chồng(h,w) = m \Leftrightarrow GTNam(h) \wedge VợChồng(h,w).$$

Nam và nữ là hai giới tính khác nhau:

$$\forall x\ GTNam(x) \Leftrightarrow \neg GTNữ(x).$$

ChaMẹ, ConCái là 2 quan hệ có tính đảo nhau:

$$\forall p,c\ ChaMẹ(p,c) \Leftrightarrow ConCái(c,p).$$

Ông bà là cha của cha một người:

$$\forall g,c\ ÔngBà(g,c) \Leftrightarrow \exists p\ ChaMẹ(g,p) \wedge ChaMẹ(p,c).$$

Anh chị em là những đứa trẻ có cùng cha mẹ:

$$\forall x,y\ AnhChịEm(x,y) \Leftrightarrow x \neq y \wedge \exists p\ ChaMẹ(p,x) \wedge ChaMẹ(p,y).$$

...

Chú ý không phải tất cả các câu logic đều là tiền đề. Một số trong chúng là định lý (có thể suy luận từ tiền đề), ví dụ, nói quan hệ anh chị em là đối xứng:

$$\forall x,y\ AnhChịEm(x,y) \Leftrightarrow AnhChịEm(y,x).$$

Câu trên là một định lý vì từ câu tiền đề "*anh chị em là những đứa trẻ có cùng cha mẹ*" chúng ta có thể suy ra điều trên. Nếu chúng ta hỏi cơ sở tri thức câu này, câu trả lời sẽ là true. Như vậy có thể thấy rằng cơ sở tri thức chỉ cần các tiền đề mà không cần các câu định lý vì nó không làm tăng tri thức hay nói chính xác là nó không làm tăng tập các câu kết luận. Tuy nhiên, các câu định lý lại giảm chi phí cho việc tính toán, suy luận. Nếu câu cần truy vấn có sẵn trong cơ sở tri thức thì chúng ta không mất thêm một số công đoạn suy luận qua hệ thống các tiền đề.

Bên cạnh đó, cũng không phải tất cả các câu tiền đề đều cần định nghĩa. Một số chỉ cung cấp thông tin để làm rõ hơn cho các vị từ chính. Ngược lại, một số vị từ lại không có định nghĩa một cách trọn vẹn. Nguyên nhân là chúng ta không có đủ đặc trưng về chúng một cách đầy đủ. Chẳng hạn, chúng ta không thể định nghĩa hoàn toàn câu:

$$\forall x \; ConNgười(x) \Leftrightarrow ...$$

Logic bậc nhất cho phép việc sử dụng các vị từ như *ConNgười* mà không cần định nghĩa rõ ràng về nó. Thay vào đó, chúng ta có thể đặc tả những điểm chính mà mỗi người có và những thuộc tính mà làm thành một con người:

$$\forall x \; ConNgười(x) \Rightarrow ...$$

$$\forall x \ldots \Longrightarrow ConNgười(x).$$

II.2 Số, tập hợp và danh sách

Số có lẽ là ví dụ hay nhất để minh họa cách xây dựng một cái khổng lồ từ một tập rất nhỏ các tiền đề. Ở đây, chúng ta xét dãy số tự nhiên hay số nguyên không âm. Gọi vị từ *SốTựNhiên* để xác định một số có là số tự nhiên không, ngoài ra còn có hằng số 0, một kí hiệu hàm *S* để chỉ số kế tiếp. Các tiền đề Peano sẽ định nghĩa số tự nhiên và phép cộng. Số tự nhiên được định nghĩa đệ quy như sau:

$SốTựNhiên(0).$

$\forall n\ SốTựNhiên(n) \Longrightarrow SốTựNhiên(S(n)).$

Nghĩa là 0 là số tự nhiên, với mỗi một n, nếu n là số tự nhiên thì S(n) là một số tự nhiên. Do đó dãy số tự nhiên sẽ là 0, S(0), S(S(0)), ... Ngoài ra, còn cần các tiền đề để giới hạn hàm S:

$\forall n\ 0 \neq S(n).$

$\forall m,n\ m \neq n \Longrightarrow S(m) \neq S(n).$

Kế tiếp chúng ta định nghĩa phép cộng trong biểu thức của hàm S:

$$\forall m\ SốTựNhiên(m) \Longrightarrow +(0,m) = m.$$

$$\forall m,n\ SốTựNhiên(m) \wedge SốTựNhiên(n) \Longrightarrow +(S(m),n) = S(+(m,n)).$$

Câu đầu mang ý nghĩa, cộng 0 vào bất kì số tự nhiên nào cũng bằng chính số đó. Kí hiệu hàm "+" có thể được viết một cách dễ đọc hơn sử dụng cách viết trung tố, nên ta viết lại câu thứ hai như sau:

$$\forall m,n\ SốTựNhiên(m) \wedge SốTựNhiên(n) \Longrightarrow (m + 1) + n = (m + n) + 1.$$

Từ phép cộng, chúng ta có thể định nghĩa thêm phép nhân bằng cách lặp lại phép cộng, định nghĩa phép lũy thừa bằng cách lặp lại phép nhân, phép chia, tập nguyên tố,...Như vậy, có thể nói toàn bộ lí thuyết số (bao gồm cả mã hóa) có thể được xây dựng từ một hằng số, một hàm, một vị từ và bốn tiền đề.

Khái niệm tập hợp cũng là một khái niệm nền tảng của toán học. Chúng ta cần xây dựng tập hợp như tập rỗng, tập con, thêm một phần tử vào tập hợp, lấy hội và giao giữa các tập hợp. Ngoài ra, chúng ta cũng muốn biết một phần tử có là thành viên của một tập hợp không, có thể phân biệt một tập hợp với tập hợp khác dựa trên những đối tượng vắng mặt không,...

Tập hợp rỗng xem như là một hằng số, kí hiệu là { }. Vị từ một ngôi, *TậpHợp*, trả về true nếu là tập hợp. Vị từ hai ngôi $x \in s$ (x là thành viên của tập s), và $s_1 \subset s_2$ (tập s_1 là tập con của tập s_2). Các hàm nhị phân $s_1 \cap s_2$ (giao của hai tập hợp), $s_1 \cup s_2$ (hợp của hai tập hợp), và $\{x|s\}$ (tập được hình thành từ thêm phần tử x đến s). Các tiền đề:

Tiền đề 1. Tập hoặc rỗng hoặc được tạo bằng cách thêm phần tử nào đó đến tập hợp:

$$\forall s\ TậpHợp(s) \Leftrightarrow (s = \{\}) \vee (\exists x, s_2\ TậpHợp(s_2) \wedge s = \{x|s_2\}).$$

Tiền đề 2. Tập hợp rỗng là tập hợp không có phần tử trong đó, hay không có cách nào để tách tập rỗng thành các tập nhỏ hơn:

$$\neg \exists x, s\ \{x|s\} = \{\}.$$

Tiền đề 3. Tập không ảnh hưởng nếu thêm một phần tử đã có vào tập hợp:

$$\forall x,s\ x \in s \Leftrightarrow s = \{x|s\}.$$

Tiền đề 4. Phần tử thuộc tập hợp khi nó đã được thêm vào tập đó. Nói cách khác, x là thành viên của s nếu và chỉ nếu s tạo bởi tập s_2 liên kết với phần tử y thì y phải là x hoặc x là phần tử thuộc s_2:

$$\forall x,s\ x \in s \Leftrightarrow [\ \exists y,s_2\ (s = \{y|s_2\} \wedge (x=y \vee x \in s_2))].$$

Tiền đề 5. Một tập là tập hợp con của tập khác nếu và chỉ nếu tất cả phần tử của tập đầu đều là phần tử của tập thứ hai:

$$\forall s_1,s_2\ s_1 \subset s_2 \Leftrightarrow (\forall x\ x \in s_1 \Rightarrow x \in s_2).$$

Tiền đề 6. Hai tập bằng nhau nếu và chỉ nếu tập này là tập con của tập kia và ngược lại:

$$\forall s_1,s_2\ (s_1 = s_2) \Leftrightarrow (s_1 \subset s_2 \wedge s_2 \subset s_1).$$

Tiền đề 7. Một phần tử nằm ở phần giao của hai tập hợp nếu và chỉ nếu nó là thành viên của cả hai tập hợp:

$$\forall x,s_1,s_2\ x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2).$$

Tiền đề 8. Một phần tử nằm ở phần hợp của hai tập hợp nếu và chỉ nếu nó là thành viên của một trong hai tập hợp:

$$\forall x, s_1, s_2 \; x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2).$$

Danh sách cũng giống như tập hợp, điểm khác biệt là danh sách thì có thứ tự, phần tử giống nhau có thể xuất hiện nhiều hơn một lần. Chúng ta có thể sử dụng thư viện Lisp của danh sách: *Nil* là một hằng để chỉ danh sách rỗng; *DanhSách, ThêmVào, PhầnĐầu, Phần Đuôi* là hàm và *TìmKiếm* là vị từ.

III.3 Một số ví dụ chuyển câu sang logic bậc nhất

- Không phải mọi người làm vườn đều thích trời nắng.

$$\neg(\forall x)\, NgườiLàmVườn(x) \Rightarrow Thích(x, TrờiNắng).$$

- Bạn có thể lừa dối ai đó cả đời.

$$(\exists x)(\forall t)\, Người(x) \wedge ThờiGian(t) \Rightarrow CóThểBịLừaDối(x,t).$$

- Bạn có thể lừa dối mỗi người trong một lúc.

$$(\forall x)(\exists t)\, Người(x) \wedge ThờiGian(t) \Rightarrow CóThểBịLừaDối(x,t).$$

(thời điểm người ta bị lừa dối có thể khác nhau)

- Bạn có thể lừa dối mọi người trong một lúc.

$$(\exists t)(\forall x)\ Người(x) \wedge ThờiGian(t) \Rightarrow CóThểBịLừaDối(x,t).$$

(tất cả mọi người đều bị lừa dối tại cùng một thời điểm)

- Bạn không thể lừa dối mọi người cả đời.

$$\neg((\forall x)(\forall t)\ Người(x) \wedge ThờiGian(t) \Rightarrow CóThểBịLừaDối(x,t)).$$

- Mọi người đều trẻ hơn bố mình

$$(\forall x) Người(x) \Rightarrow TrẻHơn(x, Bố(x)).$$

- Mọi loài nấm tím đều độc.

$$(\forall x)\ (Nấm(x) \wedge Tím(x)) \Rightarrow Độc(x).$$

- Không có nấm tím nào độc.

$$\neg(\exists x)\ Tím(x) \wedge Nấm(x) \wedge Độc(x).$$

$(\forall x)\ (Nấm(x) \wedge Tím(x)) \Rightarrow \neg Độc(x).$

- Có đúng 2 loại nấm tím độc.

$(\exists x)(\exists y) Nấm(x) \wedge Tím(x) \wedge Nấm(y) \wedge Tím(y) \wedge \neg(x=y) \wedge$

$(\forall z)\ (Nấm(z) \wedge Tím(z)) \Rightarrow ((x=z) \vee (y=z)).$

- Tuấn không cao.

  $\neg Cao(Tuấn).$

- x ở trên y, nếu x nằm ngay trên y hoặc có một chồng gồm một hay nhiều vật khác nằm trên lẫn nhau đầu tiên là x và cuối cùng là y.

  $(\forall x)(\forall y)\ ỞTrên(x,y) \Leftrightarrow (NằmTrên(x,y) \vee (\exists z)\ (NằmTrên(x,z) \wedge$

  $ỞTrên(z,y)))$.

- Ai do dự đều thua thiệt.

  $\forall x\ DoDự(x) \Rightarrow ThuaThiệt(x).$

- Không phải mọi thứ lấp lánh đều là vàng.

$$\neg \forall x\ LấpLánh(x) \Rightarrow Vàng(x).$$

- Mọi người thích Chô-cô-la

$$\forall x, Thích(x, Chô\text{-}cô\text{-}la).$$

- Vài người thích Chô-cô-la

$$\exists x,\ Thích(x, Chô\text{-}cô\text{-}la).$$

- Tất cả trẻ con đều thích Chô-cô-la

$$\forall x,\ TrẻCon(x) \Rightarrow Thích(x, Chô\text{-}cô\text{-}la).$$

- Ai cũng thích Chô-cô-la trừ khi họ dị ứng với nó

$$\forall x,\ Thích(x, Chô\text{-}cô\text{-}la) \vee DịỨng(x, Chô\text{-}cô\text{-}la).$$

$$\forall x,\ \neg DịỨng\ (x, Chô\text{-}cô\text{-}la) \Rightarrow Thích(x, Chô\text{-}cô\text{-}la).$$

- Không phải ai cũng thích Chô-cô-la

$$\neg\ (\forall x,\ Thích(x, Chô\text{-}cô\text{-}la)).$$

$\exists x, \neg Thích(x, Chô\text{-}cô\text{-}la)$.

- Không ai thích Chô-cô-la

$\neg (\exists x, Thích(x, Chô\text{-}cô\text{-}la))$.

$\forall x, \neg Thích(x, Chô\text{-}cô\text{-}la)$.

III. KẾT CHƯƠNG

Chương này giới thiệu logic bậc nhất, một ngôn ngữ biểu diễn mạnh mẽ hơn nhiều so với logic mệnh đề. Một số điểm quan trọng cần lưu ý:

- Một mô hình của logic bậc nhất được định nghĩa bởi một tập hợp các đối tượng, mối liên hệ giữa chúng và các hàm có thể được áp dụng lên chúng.
- Các ký hiệu hằng số đặt tên các đối tượng, các đối tượng vị từ đặt tên các quan hệ và các ký hiệu hàm đặt tên hàm. Một thể hiện xác định một ánh xạ từ các ký hiệu đến mô hình. Với một thể hiện và một mô hình, chân trị của một câu được xác định.
- Một câu nguyên tố bao gồm một vị từ được áp dụng cho một hoặc nhiều biểu thức; câu này đúng chỉ khi mối quan hệ được đặt tên bởi vị từ thoả giữa các đối tượng được đặt tên bởi các biểu thức. Câu phức dùng các phép liên kết tương tự logic mệnh đề và các câu được lượng hoá cho phép biểu diễn các luật tổng quát.
- Xây dựng một cơ sở tri thức trong logic bậc nhất đòi hỏi một quá trình kỹ lưỡng phân tích miền xác định, lựa chọn từ vùng và mã hoá các tiên đề theo yêu cầu để hỗ trợ các suy diễn mong muốn.

IV. THUẬT NGỮ TIẾNG ANH

Sau đây là những thuật ngữ tiếng Anh liên quan đến những khái niệm đã dùng ở chương này:

atomic sentence: câu nguyên tố

binary relation: quan hệ hai ngôi

complex sentence: câu phức

constant symbol: ký hiệu hằng số

domain: miền xác định

equality: phép bằng

existential quantifier: lượng từ tồn tại

first order logic: logic bậc nhất

function: hàm (logic)

ground term: biểu thức cơ sở

model: mô hình

nested quantifier: lượng từ lồng nhau

predicate: vị từ

property: thuộc tính

quantifier: lượng từ

term: biểu thức

tuple: bộ ba

unary relation: quan hệ một ngôi

universal quantifier: lượng từ phổ quát

Chương 6
SUY DIỄN VỚI LOGIC BẬC NHẤT

Chương 6 trình bày các *thủ tục* để trả lời cho các câu hỏi được đặt ra trong logic bậc nhất. Chương 4 đã định nghĩa khái niệm suy diễn và chỉ ra cách làm thế nào đạt được suy diễn đúng đắn và đầy đủ trong logic mệnh đề. Chương này sẽ mở rộng kết quả đó để rút ra các thuật toán có khả năng trả lời bất kì câu hỏi phát biểu bằng logic bậc nhất nào mà có thể trả lời được.

I. SUY DIỄN MỆNH ĐỀ SO VỚI BẬC NHẤT

I.1 Luật suy diễn với lượng từ

Ta hãy bắt đầu với lượng từ phổ quát. Giả sử cơ sở tri thức có một tiền đề dân gian phát biểu rằng mọi vị vua tham lam đều xấu xa:

$$\forall x\, Vua(x) \wedge ThamLam(x) \Rightarrow XấuXa(x).$$

Từ câu trên ta hoàn toàn được phép suy ra bất kì câu nào trong những câu sau:

$$Vua(Vương) \wedge ThamLam(Vương) \Rightarrow XấuXa(Vương).$$

$$Vua(Hoàng) \wedge ThamLam(Hoàng) \Rightarrow XấuXa(Hoàng).$$

$$Vua(Cha(Vương)) \wedge ThamLam(Cha(Vương)) \Rightarrow XấuXa(Cha(Vương))$$

Luật **Thể hiện phổ quát (Universal Instantiation**, gọi tắt là UI) phát biểu rằng: có thể suy diễn ra bất kì câu nào bằng cách thay thế biến bằng **biểu thức cơ sở** (từ không chứa biến). Gọi SUBST(θ, a) là kết quả khi áp dụng phép thế θ vào câu a. Vậy luật sẽ được viết là:

$$\frac{\forall v\ a}{SUBST(\{v/g\},a)}$$

với biến v bất kì và biểu thức cơ sở g. Ví dụ, ba câu suy diễn ban đầu thu được từ các phép thế $\{x/Vương\}$, $\{x/Hoàng\}$ và $\{x/Cha(Vương)\}$.

Luật **Thể hiện tồn tại** tương ứng với lượng từ tồn tại tương đối phức tạp hơn. Với câu α bất kì, biến v và kí hiệu hằng k chưa từng xuất hiện trong cơ sở tri thức:

$$\frac{\exists v\ \alpha}{SUBST(\{v/k\},\alpha)}$$

Ví dụ từ câu:

$\exists x\ VươngMiện(x) \wedge TrênĐầu(x,Vương)$

ta có thể suy ra câu:

$VươngMiện(C_1) \wedge TrênĐầu(C_1,Vương)$

miễn sao C_1 chưa xuất hiện trong cơ sở tri thức. Về cơ bản, câu chứa lượng từ tồn tại nói rằng có một vài đối tượng thỏa mãn một điều kiện nào đó và quá trình tạo thể hiện chỉ đặt tên cho đối tượng. Đương nhiên tên này không được thuộc về đối tượng khác trước đó. Trong logic, tên mới được gọi là **hằng Skolem**. Thể hiện tồn tại là trường hợp đặc biệt của một quá trình tổng quát hơn gọi là **Skolem hóa**, sẽ được đề cập trong phần 6.V.

Thể hiện tồn tại phức tạp hơn và đóng vai trò hơi khác so với thể hiện phổ quát trong suy diễn. Trong khi Thể hiện phổ quát có thể được áp dụng nhiều lần để tạo ra nhiều hệ quả khác nhau, Thể hiện tồn tại chỉ áp dụng một lần và có thể bỏ đi câu chứa lượng từ tồn tại. Ví dụ, một khi đã thêm vào câu *Giết*(*TộiPhạm*, *NạnNhân*), ta không cần câu: $\exists x$ Giết(x, *NạnNhân*). Phát biểu một cách nghiêm ngặt, cơ sở tri thức mới không tương đương logic với cơ sở tri thức cũ, nhưng nó có thể được chứng minh là **tương đương về mặt suy diễn** theo nghĩa nó thỏa mãn được khi cơ sở tri thức ban đầu là thỏa mãn được.

I.2 Rút gọn về suy diễn mệnh đề

Khi đã có luật suy diễn cho câu không có lượng từ từ câu có lượng từ, ta có thể rút gọn suy diễn bậc nhất về suy diễn mệnh đề.

Ý tưởng đầu tiên là, cũng như câu chứa lượng từ tồn tại có thể thay thế bằng một thể hiện, câu chứa lượng từ phổ quát có thể được thay bằng tập hợp tất cả các thể hiện có thể có. Ví dụ, giả sử cơ sở tri thức chứa các câu:

$$\forall x\, Vua(x) \wedge ThamLam(x) \Rightarrow XấuXa(x)$$

$$Vua(Vương)$$

$$ThamLam(Vương)$$

$$AnhEm(Hoàng, Vương) \qquad (6.1)$$

Tiếp theo, áp dụng luật thể hiện phổ quát vào câu đầu tiên, sử dụng mọi phép thế biểu thức cơ sở có thể có từ tập từ vựng của cơ sở tri thức, trong trường hợp này là $\{x/Vương\}$ và $\{x/Hoàng\}$. Ta được:

$$Vua(Vương) \wedge ThamLam(Vương) \Rightarrow XấuXa(Vương)$$

$$Vua(Hoàng) \wedge ThamLam(Hoàng) \Rightarrow XấuXa(Hoàng)$$

và loại bỏ câu chứa lượng từ phổ quát. Bây giờ, cơ sở tri thức về mặt cơ bản chứa các mệnh đề nếu như ta xem những câu nguyên tử như $Vua(Vương)$, $ThamLam(Vương)$ như kí hiệu mệnh đề. Do đó có thể áp dụng bất kì thuật toán mệnh đề nào trong Chương 4 để rút ra kết luận, chẳng hạn như $XấuXa(Vương)$.

Người ta có thể làm cho kĩ thuật **mệnh đề hóa** trở nên hoàn toàn tổng quát tức là mọi cơ sở tri thức và truy vấn bậc nhất đều có thể được mệnh đề hóa sao cho trật tự suy diễn được bảo toàn. Như vậy, ta có một thủ tục quyết định hoàn chỉnh để suy diễn...hoặc có thể không. Vấn đề đặt ra là: khi cơ sở tri thức chứa kí hiệu hàm, tập phép thế biểu thức cơ sở sẽ trở nên vô hạn! Ví dụ, nếu cơ sở tri thức chứa kí hiệu Cha thì có vô số từ như $Cha(Cha(Cha(Vương)))$ được tạo ra. Thuật toán mệnh đề sẽ gặp khó khăn với tập câu lớn vô hạn như thế. Vì lý do trên, suy diễn trong logic bậc nhất là bán quyết định, tức là các thuật

toán trả lời *có* với mọi câu suy ra được nhưng không có thuật toán nào trả lời *không* với câu không suy ra được.

II. PHÉP ĐỒNG NHẤT

Phần trước đã mô tả cách suy diễn bậc nhất tồn tại cho đến đầu những năm 1960, tuy nhiên việc mệnh đề hoá là không khả thi. Ví dụ, cho truy vấn *XấuXa*(x) và cơ sở tri thức như ở phần 6.I, thì việc phát sinh những câu như *Vua*(*Hoàng*) $\wedge$ *ThamLam*(*Hoàng*) $\Rightarrow$ *XấuXa*(*Hoàng*) là quá cứng nhắc và không cần thiết. Phần tiếp theo xử lý cho trường hợp này.

II.1 Luật suy diễn bậc nhất

Suy diễn "Vương thì xấu xa" sẽ thực hiện như sau: tìm một vài x sao cho x là vua và x tham lam thì suy ra x xấu xa. Tổng quát hơn, nếu có một số phép thế θ làm cho tiền đề của câu kéo theo giống với các câu đã có trong cơ sở tri thức, thì ta có thể xác nhận phần kết luận, sau khi áp dụng θ. Trong trường hợp này, phép thế {x/*Vương*} đã đạt được mục đích đó. Ta còn có thể làm nhiều việc hơn trong bước suy diễn. Giả sử thay vì biết *ThamLam*(*Vương*), ta biết *mọi người* đều tham lam:

$\forall y$*ThamLam*(y) (6.2)

Như thế vẫn có thể kết luận *XấuXa*(*Vương*) vì ta đã biết Vương là vua (cho trước) và Vương tham lam (vì mọi người đều tham lam). Điều cần làm là tìm một phép thế sao cho biến trong câu kéo theo và các câu đơn khớp nhau. Trong trường hợp này, áp dụng phép thế {x/*Vương*, y/*Vương*} vào tiền đề*Vua*(x) và *ThamLam*(x) và câu trong cơ sở tri thức *Vua*(*Vương*) và*ThamLam*(y) sẽ làm cho chúng giống nhau. Do đó, ta có thể suy ra phần kết luận.

Quá trình suy diễn có thể tóm gọn bằng một luật suy diễn, gọi là **Tam đoạn luận tổng quát**: Với mọi câu nguyên tử: p_i, p_i', q và phép thế θ sao cho SUBST(θ, p_i') = SUBST(θ, p_i), thì với mọi i,

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{SUBST(\theta, q)}$$

Có n+1 tiền đề trong luật này: n câu nguyên tử p_i' và một câu kéo theo. Kết luận là kết quả của việc áp dụng phép thế θ vào hệ quả q. Ví dụ:

p_1' là *Vua*(*Vương*) p_1 là *Vua*(x)

p_2' là *ThamLam*(y) p_2 là *ThamLam*(x)

θ là {x/*Vương*, y/*Vương*} q là *XấuXa*(x)

SUBST(θ, q) là *XấuXa*(*Vương*)

Dễ dàng chứng minh được *Tam đoạn luận Tổng quát* là luật suy diễn đúng đắn. Đầu tiên, ta quan sát thấy với câu p bất kì (có các biến được giả sử chứa lượng từ phổ quát) và với phép thế θ bất kì:

$$p \models SUBST(\theta, p)$$

Phát biểu này tồn tại vì những lí do giống như của luật Thể hiện tổng quát. Nó tồn tại trong trường hợp cụ thể với một θ thỏa các điều kiện của luật tam đoạn luận tổng quát. Do đó, từ p_1',..., p_n', ta có thể suy ra:

$$SUBST(\theta, p_1') \wedge .... \wedge SUBST(\theta, p_n')$$

và từ: $p_1 \wedge \ldots \wedge p_2 \Rightarrow q$, ta có:

$$SUBST(\theta, p_1) \wedge .... \wedge SUBST(\theta, p_n) \Rightarrow SUBST(\theta, q)$$

Bây giờ, θ trong tam đoạn luận tổng quát được định nghĩa sao cho SUBST(θ, p_i') = SUBST(θ, p_i) với mọi i; do đó câu đầu tiên trong số hai câu trùng khớp hoàn toàn với tiền đề của câu thứ hai. Vậy, *SUBST*(θ, q) tuân theo tam đoạn luận.

II.2 Đồng nhất

Luật suy diễn mở rộng đòi hỏi tìm các phép thế sao cho các biểu thức logic trở nên giống nhau. Quá trình này gọi là **đồng nhất** và là một bộ phận quan trọng trong mọi thuật toán suy diễn bậc nhất. Thuật toán ĐồngNhất nhận vào hai câu và trả ra **hàm đồng nhất** nếu có:

ĐồngNhất(p, q) = θ khi SUBST(θ, p) = SUBST(θ, q)

Chúng ta hãy xem một vài ví dụ về cách vận hành của ĐồngNhất. Giả sử có truy vấn *Biết*(*Vương*, *x*): Vương biết ai? Ta có thể trả lời cho truy vấn này bằng cách tìm mọi câu trong cơ sở tri thức có thể đồng nhất với *Biết*(*Vương*, *x*). Dưới đây là các kết quả đồng nhất với bốn câu khác nhau thuộc cơ sở tri thức:

ĐồngNhất(*Biết*(*Vương*, *x*), *Biết*(*Vương*, *Công*)) = {*x*/*Công*}

ĐồngNhất(*Biết*(*Vương*, *x*), *Biết*(*y*, *Hoàng*)) = {*x*/*Hoàng*, *y*/*Vương*}

ĐồngNhất(*Biết*(*Vương*, *x*), *Biết*(*y*, *Mẹ*(*y*))) = {*y*/*Vương*, *x*/*Mẹ*(*Vương*)}

ĐồngNhất(*Biết*(*Vương*, *x*), *Biết*(*x*, *Công*)) = **Fail**

Phép đồng nhất cuối cùng thất bại vì *x* không thể nhận giá trị *Vương* và *Công* cùng một lúc. Lưu ý rằng *Biết*(*x*, *Công*) có nghĩa là "Mọi người biết Công", do đó có thể suy ra *Vương* biết *Công*. Vấn đề xảy ra chỉ vì hai câu dùng cùng một tên biến *x*. Ta tránh điều này bằng cách **phân chia chuẩn hóa** một trong hai câu đang được đồng nhất, tức là đổi tên các biến để tránh đụng độ. Ví dụ, ta có thể đổi *x* trong *Biết*(*x*, *Công*) thành z_{17} (tên biến mới) mà không làm thay đổi ý nghĩa. Bây giờ phép đồng nhất sẽ thực hiện được:

ĐồngNhất(*Biết*(*Vương*, *x*), *Biết*(z_{17}, *Công*)) = {*x*/*Công*, z_{17}/*Vương*}

Ở đây có thêm một điều phức tạp: ĐồngNhất cần trả về một phép thế sao cho hai đối số giống nhau. Nhưng có thể có hơn một hàm đồng nhất. Ví dụ, ĐồngNhất(*Biết*(*Vương*, *x*), *Biết*(*y*, *z*)) trả về {*y*/*Vương*, *x*/*z*} hoặc {*y*/*Vương*, *x*/*Vương*, *z*/*Vương*}. Hàm đồng nhất đầu tiên cho *Biết*(*Vương*, *z*} làm kết quả đồng nhất, trong khi hàm đồng nhất thứ hai cho ra *Biết*{*Vương*/*Vương*}. Kết quả thứ hai có thể rút ra từ cái thứ nhất bằng cách thế {*z*/*Vương*}. Ta nói hàm đồng nhất đầu tiên tổng quát hơn vì nó đặt ít ràng buộc hơn về giá trị biến. Như vậy, với mỗi cặp biểu thức có thể đồng nhất được, có một hàm đồng nhất tổng quát nhất (gọi tắt là MGU) với cách thay tên biến duy nhất. Trong trường hợp này là {*y*/*Vương*, *x*/*z*}.

Hình 6.1 trình bày thuật toán tính các MGU. Qui trình rất đơn giản: khảo sát đệ qui đồng thời hai biểu thức, trong lúc đó xây dựng hàm đồng nhất, nếu hai điểm tương ứng trong cấu trúc không khớp nhau thì thất bại. Ở đây có một bước tốn nhiều chi phí: khi so một biến với một từ phức hợp thì phải kiểm tra xem bản thân biến đó có xuất

hiện trong từ hay không, nếu có thì so khớp thất bại vì không thể xây dựng hàm đồng nhất nhất quán. Động tác này gọi là **kiểm tra xuất hiện** làm cho độ phức tạp của toàn bộ thuật toán là bậc hai của kích thước các biểu thức cần đồng nhất. Một vài hệ thống, bao gồm mọi hệ thống lập trình logic, bỏ qua kiểm tra xuất hiện và đôi khi cho ra kết quả lập luật không đúng đắn; những hệ thống khác sử dụng các thuật toán phức tạp hơn với độ phức tạp thời gian tuyến tính.

```
function ĐồngNhất(x, y, θ) return phép thế làm cho x và y đồng nhất
  input:  x, một biến, hằng, danh sách hay tổ hợp
          y, một biến, hằng, danh sách hay tổ hợp
          θ, phep thế được xây dựng (tuỳ chọn, thường là rỗng)
  If θ = thất bại then return thất bại.
  else if x = y then return θ
  else if Biến?(x) then return ĐồngNhấtBiến(x, y, θ)
  else if Biến?(y) then return ĐồngNhấtBiến(y, x, θ)
  else if HỗnHợp?(x) and HỗnHợp?(y) then
       return ĐồngNhất(ARG[x], ARG[y], ĐồngNhất(OP[x], OP[y],
θ))
  else if DanhSách?(x) and DanhSách?(y) then
       return ĐồngNhất(REST[x], REST[y], ĐồngNhất(FIRST[x],
FIRST[y], θ))
  else return thất bại

function ĐồngNhấtBiến(var, y, θ) return một phép thế
  input:  var, một biến
          x, một biểu thức
          θ, phép thế được xây dựng
  if {var/val} ∈ θ then return ĐồngNhất(val, x, θ)
  else if {x/val} ∈ θ then return ĐồngNhất(var, val, θ)
  else if KiểmTraXuấtHiện?(var, x) then return thất bại
  else return thêm {var/x} vào θ
```

Hình 6.1 - *Thuật toán đồng nhất.*

III. SUY DIỄN TIẾN

Thuật toán suy diễn tiến đã được giới thiệu trong logic mệnh đề. Ý tưởng đơn giản như sau: bắt đầu bằng các câu nguyên tử trong cơ sở

tri thức và áp dụng tam đoạn luận theo hướng thẳng tiến, thêm câu nguyên tử mới, cho đến khi không đưa ra được suy diễn nào nữa.

III.1 Các mệnh đề xác định bậc nhất

Mệnh đề xác định bậc nhất rất giống mệnh đề xác định dạng mệnh đề: chúng là tuyển của các từ trong đó chỉ có một từ dương. Mệnh đề xác định là nguyên tử hoặc là câu kéo theo có tiền đề là tuyển các từ dương và hệ quả là một từ dương đơn nhất. Đây là các mệnh đề xác định bậc nhất:

$$Vua(x) \wedge ThamLam(x) \Rightarrow XấuXa(x)$$

$$Vua(Vương)$$

$$ThamLam(y)$$

Không như từ mệnh đề, từ bậc nhất có thể chứa biến và các biến được giả sử là chứa lượng từ phổ quát. (Thông thường, ta bỏ lượng từ phổ quát khi viết mệnh đề xác định.).Mệnh đề xác định là dạng chuẩn hóa phù hợp cho việc sử dụng Tam đoạn luận tổng quát.

Không phải mọi cơ sở dữ liệu đều có thể chuyển thành tập mệnh đề xác định do ràng buộc về từ đơn dương. Hãy xét bài toán sau:

Luật nói rằng: người Mỹ bán vũ khí cho nước thù địch là có tội. Nước Nono, kẻ thù của Mỹ, có một số tên lửa, và mọi tên lửa này đều do Đại tá West bán, ông này là người Mỹ.

Ta sẽ chứng minh West là tội phạm. Đầu tiên, biểu diễn các sự kiện này dưới dạng mệnh đề xác định bậc nhất. Phần tiếp theo sẽ chỉ ra cách mà thuật toán suy diễn tiến giải quyết vấn đề

"...người Mỹ bán vũ khí cho nước thù địch là có tội":

$$NgườiMỹ(x) \wedge VũKhí(y) \wedge Bán(x, y, z) \wedge ThùĐịch(z) \Rightarrow TộiPhạm(x) \quad (6.3)$$

"Nono....có một vài tên lửa." Câu $Có(Nono, x) \wedge TênLửa(x)$ được chuyển thành hai mệnh đề xác định bằng phép loại lượng từ tồn tại, đưa ra hằng số mới M_1:

$$Có(Nono, M_1) \quad (6.4)$$

$$TênLửa(M_1) \tag{6.5}$$

"Mọi tên lửa này đều do Đại tá West bán cho":

$$TênLửa(x) \wedge Có(Nono, x) \Rightarrow Bán(West, x, Nono) \tag{6.6}$$

Ta cũng cần biết tên lửa là vũ khí:

$$TênLửa(x) \Rightarrow VũKhí(x) \tag{6.7}$$

và ta biết rằng kẻ thù của nước Mỹ cũng tính là "nước thù địch":

$$KẻThù(x, NgườiMỹ) \Rightarrow ThùĐịch(x) \tag{6.8}$$

"West là người Mỹ...":

$$NgườiMỹ(West) \tag{6.9}$$

"Nước Nono, kẻ thù của Mỹ":

$$KẻThù(Nono, NgườiMỹ) \tag{6.10}$$

Cơ sở tri thức không chứa kí hiệu hàm và do đó là một thể hiện của lớp các cơ sở tri thức **Datalog** – tức là, tập hợp các mệnh đề xác định bậc nhất không có kí hiệu hàm.

III.2 Thuật toán suy diễn tiến đơn giản

Thuật toán suy diễn tiến đầu tiên mà ta xem xét là một thuật toán rất đơn giản, xem Hình 6.2. Khởi đầu từ những sự kiện đã biết, thuật toán kích hoạt mọi luật có tiền đề được thỏa và thêm kết luận của chúng vào nhóm sự kiện đã biết. Quá trình lặp đi lặp lại cho đến khi trả lời được câu truy vấn (giả sử rằng chỉ cần một lời giải) hoặc không có sự kiện mới nào được thêm vào. Chú ý rằng sự kiện không phải là "mới" nếu nó thay đổi tên của một sự kiện đã biết. Một câu gọi là đổi tên của câu khác nếu chúng hoàn toàn giống nhau ngoại trừ tên biến. Ví dụ, *Thích*(*x*, *Kem*) và *Thích*(*y*, *Kem*) là đổi tên của nhau vì chúng chỉ khác *x* và *y*; ý nghĩa của chúng là như nhau: mọi người đều thích kem.

Dưới đây là ví dụ minh họa cách thuật toán suy diễn tiến FOL-FC-ASK hoạt động với bài toán tội phạm. Các câu kéo theo là (6.3), (6.6), (6.7) và (6.8). Ta cần hai vòng lặp:

```
function FOL-FC-ASK(KB, a) return một phép thế hoặc false
  input:  KB, cơ sở tri thức, một tập các mệnh đề hữu hạn bậc nhất
          a, câu truy vấn, một câu nguyên tố
  local variable: new, câu mới được suy dẫn ra trong mỗi lần lặp
  repeat until new là rỗng
     new ← { }
     for each câu r in KB do
        (p1 ∧ ... ∧ pn ⇒ q) ← STANDARDIZE-APART(r)
        for each θ sao cho SUBST(θ, p1 ∧ ... ∧ pn) = SUBST(θ, p'1 ∧ ...
                                   ∧ p'n) với p'1,..., p'n trong KB
           q' ← SUBST(θ, q)
           if q' không là một câu được đổi tên từ các câu đã có trong KB
                                                  hay new then do
              thêm q' vào new
              φ ← ĐồngNhất(q', a)
              if φ không thất bại then return φ
        thêm new vào KB
  return false
```

Hình 6.2 - *Thuật toán suy diễn tiến. Trong mỗi vòng lặp, thuật toán thêm vào KB mọi câu nguyên tử có thể suy ra được trong một bước từ câu suy dẫn và những câu nguyên tử đã có sẵn trong KB.*

- Trong vòng lặp thứ nhất, luật (6.3) có tiền đề không thỏa.

  Luật (6.6) thỏa với $\{x/M_1\}$ và $Bán\{West, M_1, Nono\}$ được thêm vào.

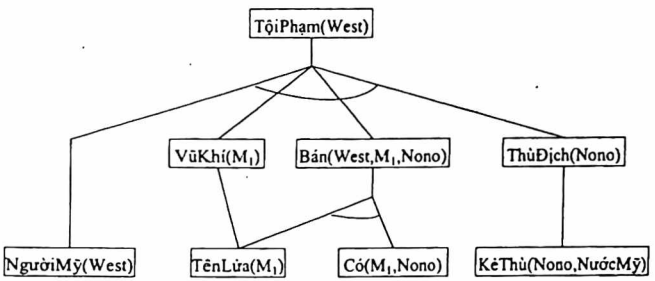
  Luật (6.7) thỏa với $\{x/M_1\}$ và $VũKhí(M_1)$ được thêm vào.

  Luật (6.8) thỏa với $\{x/Nono\}$ và $ThùĐịch(Nono)$ được thêm vào.

- Trong vòng lặp thứ hai, luật (6.3) thỏa mãn với $\{x/West, y/M_1, z/Nono\}$ và $TộiPhạm(West)$ được thêm vào.

Hình 6.3 trình bày cây chứng minh được phát sinh. Chú ý rằng không có suy luận mới nào có thể có ở điểm này vì mọi câu có thể kết luận được bằng suy diễn tiến đã nằm sẵn trong KB. Một cơ sở tri thức như thế gọi là **điểm cố định** của quá trình suy diễn. Điểm cố định đạt được bằng suy diễn tiến với mệnh đề xác định bậc nhất tương tự như trong

suy diễn tiến mệnh đề; điểm khác biệt chủ yếu là điểm cố định bậc nhất có thể bao gồm cả câu nguyên tử ngầm chứa lượng từ phổ quát.



Hình 6.3 - *Cây chứng minh được phát sinh bằng suy diễn tiến trên ví dụ tội phạm. Các sự kiện ban đầu ở tầng dưới cùng, sự kiện suy ra trong vòng lặp đầu tiên ở tầng giữa, và các sự kiện suy ra từ vòng lặp thứ hai ở tầng trên cùng.*

Thuật toán suy diễn tiến đối với logic bậc nhất là đầy đủ vời vì mỗi suy diễn đều là sự áp dụng của tam đoạn luận tổng quát. Thuật toán này cũng đầy đủ đối với các cơ sở tri thức mệnh đề xác định. Đối với các mệnh đề xác định tổng quát với các ký hiệu hàm, thuật toán có thể phát sinh vô hạn các sự kiện mới, do đó ta cần cẩn thận hơn. Đối với logic bậc nhất tổng quát, suy diễn với các mệnh đề xác định là bán quyết định.

IV. SUY DIỄN LÙI

Một nhóm lớn các thuật toán suy diễn logic sử dụng hướng tiếp cận **suy diễn lùi**. Những thuật toán này đi lùi từ câu hỏi đích, suy luận thông qua các luật để tìm sự kiện đã biết củng cố cho phép chứng minh. Phần này mô tả thuật toán cơ bản, sau đó trình bày cách sử dụng nó trong **lập trình logic**, đây là dạng suy diễn tự động được sử dụng rộng rãi nhất.

IV.1 Thuật toán suy diễn lùi

Hình 6.4 trình bày một thuật toán suy diễn lùi đơn giản, FOL-BC-ASK. Nó được gọi với một danh sách các câu hỏi đích chứa trong một phần tử đơn, là câu truy vấn ban đầu, và trả về tập các phép thế thỏa truy vấn. Có thể xem danh sách câu hỏi đích này như một "ngăn xếp" đang đợi để được xử lí; nếu tất cả đều được thỏa thì nhánh hiện tại của quá trình chứng minh thành công. Thuật toán lấy câu hỏi đích đầu tiên trong danh sách và tìm mọi mệnh đề trong cơ sở tri thức có từ dương, hay phần đầu, đồng nhất được với câu hỏi đích. Mỗi mệnh đề như thế tạo ra lời gọi đệ qui mới và phần tiền đề, hay phần thân, của mệnh đề được thêm vào ngăn xếp chứa câu hỏi đích. Lưu ý rằng sự kiện là mệnh đề có phần đầu nhưng không có phần thân, nên khi một câu hỏi đích đồng nhất được với sự kiện đã biết, không có câu hỏi đích mới được thêm vào ngăn xếp và câu hỏi đích được giải quyết. Hình 6.5 là cây chứng minh cho việc suy dẫn *TộiPhạm(West)* từ câu 6.3 đến 6.10.

function FOL-BC-ASK(*KB*, *truy-vấn*, θ) **return** một phép thế hay *false*
  input: **KB**, cơ sở tri thức
    truy-vấn, một danh sách truy vấn dạng nối liền (θ đã được áp dụng)
    θ, phép thế hiện tại, ban đầu là phép thế rỗng{}
  local variable: *trả-lời*, một tập phép thế, khởi tạo rỗng

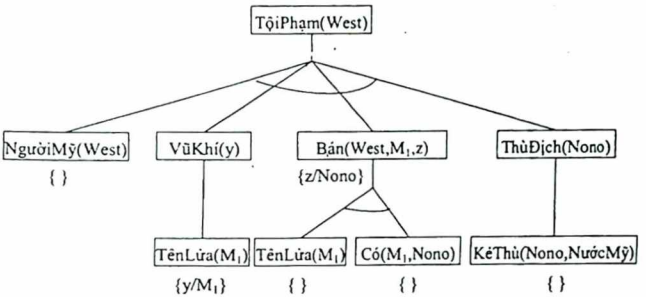
  if *truy-vấn* rỗng **then return** {θ}
  q' ← SUBST(θ, CâuĐầu(*truy-vấn*))
  for each câu r **in** *KB* trong đó CácThànhPhầnChuẩnHoá(r) = $(p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ và θ' ← ĐồngNhất(q, q') thành công
    truy-vấn-mới ← [$p_1, \ldots, p_n$|PhầnCònLại(*truy-vấn*)]
    trả-lời ← FOL-BC-ASK(*KB*, *truy-vấn-mới*, TổngHợp(θ',θ)) ∪ *trả-lời*
  return *trả-lời*

Hình 6.4 - *Thuật toán suy diễn lùi đơn giản*

Thuật toán sử dụng tổng hợp các phép thế. Hàm TổngHợp(θ_1, θ_2) là phép thế mà hiệu quả của nó giống như việc áp dụng lần lượt từng phép thế. Tức là:

$$\text{SUBST}(\text{TổngHợp}(\theta_1, \theta_2), p) = \text{SUBST}(\theta_2, \text{SUBST}(\theta_1, p)).$$

Trong thuật toán, phép kết biến hiện tại, lưu trong θ, được tổng hợp với các phép kết có được từ việc đồng nhất câu hỏi đích với phần đầu của mệnh đề, cho ra tập các phép kết hiện thời mới cho lời gọi đệ qui.



Hình 6.5 - *Cây chứng minh được xây dựng bằng suy diễn lùi để chứng minh West là tội phạm. Cần đọc cây từ dưới lên, trái qua phải. Để chứng minh TộiPhạm(West), ta phải chứng minh 4 thành phần bên dưới nó. Một số trong đó thuộc cơ sở tri thức và những cái khác đòi hỏi phải suy diễn lùi tiếp. Phép kết cho mỗi lượt đồng nhất thành công được trình bày kế mỗi "câu hỏi đích phụ" tương ứng. Lưu ý rằng: khi một "câu hỏi đích phụ" trong một câu hội thành công, phép thế của nó được áp dụng cho các "câu hỏi đích phụ" tiếp theo. Do đó, lúc FOL-BC-ASK đi đến phần hội cuối cùng, ThùĐịch(z) đã kết với Nono.*

Suy diễn lùi là một thuật toán tìm kiếm theo chiều sâu. Điều này có nghĩa là nó cần không gian tuyến tính với kích thước của phép chứng minh (không gian cần để tích lũy đáp án). Đồng thời suy diễn lùi cũng gặp vấn đề trạng thái lặp lại và không đầy đủ (không giống như suy diễn tiến). Phần tiếp theo thảo luận vấn đề này và một số giải pháp, nhưng đầu tiên, hãy xem suy diễn lùi được sử dụng như thế nào trong các hệ thống lập trình logic.

IV.2 Lập trình logic

Lập trình logic là kỹ thuật hiện thực hoá ý tưởng: Các hệ thống nên được xây dựng bằng cách biểu diễn tri thức theo ngôn ngữ hình thức và các bài toán được giải bằng cách thực hiện các qui trình suy diễn trên tri thức này. Ý tưởng được tóm tắt bằng công thức của Robert Kowalski:

Thuật toán = Logic + Điều khiển.

Prolog là ngôn ngữ lập trình logic được sử dụng rộng rãi nhất cho đến nay, có đến hàng trăm ngàn người sử dụng. Nó được dùng chủ yếu như một ngôn ngữ viết nguyên mẫu nhanh và để biểu diễn kí hiệu các tác vụ như viết trình biên dịch và dịch ngôn ngữ tự nhiên. Nhiều hệ chuyên gia được viết bằng Prolog cho các lĩnh vực như luật pháp, y khoa, tài chính và nhiều ngành khác.

Chương trình Prolog là các tập mệnh đề xác định được viết theo kí hiệu có phần hơi khác với dạng bậc nhất chuẩn. Prolog sử dụng chữ cái viết hoa cho biến và chữ cái viết thường cho hằng. Các mệnh đề có phần đầu nằm trước thân, ":-" dùng để biểu diễn phép suy ra trái, dấu phẩy chia cắt các từ trong thân và dấu chấm báo hiệu hết câu.

```
toipham(X):-
nguoimy(X),vukhi(Y),ban(X,Y,Z),thudich(Z).
```

Prolog hỗ trợ kí hiệu danh sách và đại số. Ví dụ, chương trình Prolog `append(X,Y,Z)` sẽ thành công nếu danh sách Z là kết quả nối danh sách X và Y:

```
append([],Y,Y).
append([A|X],Y,[A|Z]) :- append(X,Y,Z).
```

Các mệnh đề này có ý nghĩa như sau: (1) là nối một danh sách rỗng với danh sách Y tạo ra cùng một danh sách Y và (2) `[A|Z]` là kết quả của việc nối `[A|X]` lên trên Y, với Z là kết quả của việc nối X lên trên Y. Ta ta có thể truy vấn `append(A,B,[1, 2])`: hai danh sách nào nối với nhau cho ra [1, 2]? Ta nhận được các giải pháp sau:

```
A = [ ]        B = [1,2]
```

```
A = [1]          B = [2]
A = [1,2]        B = [ ]
```

Chương trình Prolog được thực thi thông qua suy diễn lùi theo chiều sâu, các mệnh đề được thử theo thứ tự đã khai báo trong cơ sở tri thức. Một vài khía cạnh của Prolog vượt ra khỏi suy diễn logic chuẩn:

Có một tập hợp các hàm xây dựng sẵn cho phép toán đại số. Từ chứa kí hiệu hàm được "chứng minh bằng chạy mã lệnh hơn là thực hiện suy diễn". Ví dụ, "X bằng 4+3" thành công khi X kết với 7. Mặt khác "5 bằng X + Y" thất bại vì hàm xây dựng sẵn không giải phương trình tùy ý.

Nhiều vị từ xây dựng sẵn có hiệu ứng phụ khi thực thi. Chúng bao gồm vị từ input-output và vị từ assert/retract để chỉnh cơ sở tri thức. Những vị từ này không tồn tại trong logic và có thể gây ra nhiều hiệu ứng khó hiểu, ví dụ như trường hợp các sự kiện được xác nhận tại một nhánh của cây chứng minh mà cuối cùng lại thất bại.

Prolog cho phép dạng phủ định gọi là **phủ định khi thất bại**. Một câu hỏi đích phủ định `not P` được xem là đã chứng minh nếu hệ thống không thể chứng minh *P*. Do đó, câu:

```
song(X) :- not chet(X).
```

có thể đọc là "Mọi người đều sống nếu không chứng minh được là chết".

Prolog có toán tử bằng: " = ", nhưng nó không có đầy đủ năng lực như phép bằng logic. Một câu hỏi đích chứa toán tử bằng thành công nếu hai toán hạng đồng nhất được và ngược lại thì thất bại. Vậy `X + Y = 2 + 3` thành công với X là 2 và Y là 3, nhưng `sao_hôm = sao_mai` thất bại. (Trong logic cổ điển, đẳng thức thứ hai có thể đúng hoặc có thể sai). Không có sự kiện hay luật nào về phép bằng có thể đảm bảo.

Thuật toán đồng nhất của Prolog bỏ kiểm tra xuất hiện. Điều này có nghĩa là một vài suy diễn không đúng đắn có thể xảy ra, sẽ không có gì nghiêm trọng trừ khi dùng Prolog để chứng minh định lý toán học.

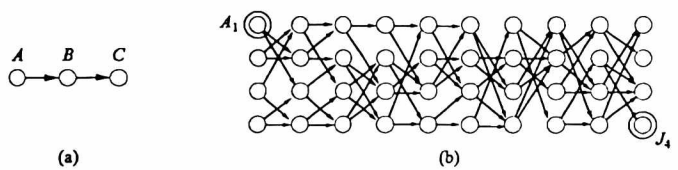
IV.3 Suy diễn dư thừa và lặp vô tận

Phần này thảo luận đến điểm yếu của Prolog: Sự sai lệch giữa tìm kiếm theo chiều sâu và cây tìm kiếm khi có trạng thái lặp và đường đi vô tận. Xét chương trình logic sau có nhiệm vụ quyết định có đường đi giữa hai đỉnh trên một đồ thị có hướng hay không:

```
path(X,Z) :- link(X,Z).
path(X,Z) :- path(X,Y),link(Y,Z).
```

Hình 6.6(a) vẽ một đồ thị ba nút đơn giản được mô tả bằng các sự kiện `link(a,b)` và `link(b,c)`. Với chương trình trên, truy vấn `path(a,c)` phát sinh cây tìm kiếm như trong Hình 6.7(a). Mặt khác, nếu ta đặt hai mệnh đề theo thứ tự:

```
path(X,Z) :- path(X,Y),link(Y,Z).
path(X,Z) :- link(X,Z).
```

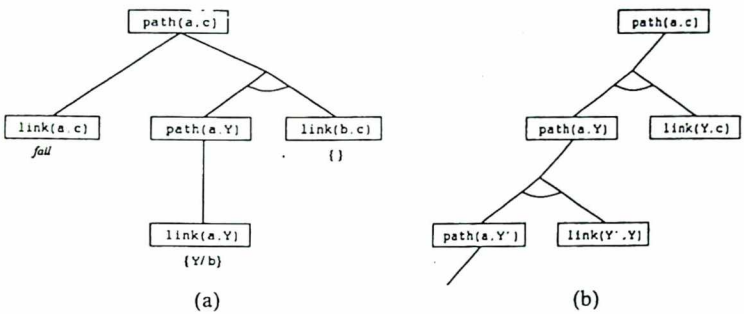


Hình 6.6 - *(a) Tìm một đường đi từ A đến C có thể làm cho Prolog lặp vô hạn. (b) Đồ thị có mỗi nút được nối với hai con bất kì trong tầng kế tiếp. Tìm đường đi từ A_1 đến J_4 cần 877 phép suy diễn.*

thì Prolog sẽ theo con đường vô tận như trong Hình 6.9(b). Như vậy, Prolog là bộ chứng minh định lí không đầy đủ cho các mệnh đề xác định - ngay cả với Datalog, như ví dụ đã cho thấy - vì đối với một số cơ sở tri thức, nó không thể chứng minh được các câu vốn có thể suy ra được. Lưu ý rằng suy diễn tiến không gặp vấn đề này: một khi suy ra được `path(a,b)`, `path(b,c)` và `path(a,c)` thì suy diễn tiến dừng.

Suy diễn lùi sâu dần cũng gặp vấn đề tính toán dư thừa. Ví dụ, khi tìm đường đi từ A_1 đến J_4 trong Hình 6.6(b), Prolog thực hiện 877 suy

diễn, hầu hết trong số đó tìm đường đi đến các nút không đến được câu hỏi đích. Điều này tương tự như bài toán trạng thái lặp trong Chương 1. Tổng số suy diễn có thể là hệ số mũ so với số sự kiện nền được phát sinh. Nếu ta áp dụng suy diễn tiến, tối đa n^2`path(X,Y)` được phát sinh, nối n nút với nhau. Với bài toán trong Hình 6.6(b), chỉ cần 62 suy diễn là đủ.



Hình 6.7 - *(a) Chứng minh tồn tại đường đi từ A đến C. (b) Cây chứng minh vô hạn được phát sinh khi các mệnh đề nằm "sai" thứ tự.*

V. HỢP GIẢI

Nhóm cuối cùng trong ba nhóm hệ thống logic là dựa trên **hợp giải**. Chương 4 đã trình bày hợp giải mệnh đề là một thủ tục suy diễn bằng chứng minh phản chứng đầy đủ dành cho logic mệnh đề. Phần này sẽ mở rộng hợp giải cho logic bậc nhất.

V.1 Dạng hội chuẩn đối với logic bậc nhất

Cũng như dạng mệnh đề, hợp giải bậc nhất đòi hỏi các câu phải ở dạng hội chuẩn CNF – tức là hội các mệnh đề, mỗi mệnh đề gồm nhiều từ. Từ có thể chứa biến, giả sử đã được phổ quát hóa. Ví dụ, câu:

$\forall x$ $NgườiMỹ(x) \wedge VũKhí(y) \wedge Bán(x, y, z) \wedge ThùĐịch(z) \Rightarrow TộiPhạm(x)$.

chuyển sang CNF sẽ là:

¬NgườiMỹ(x) ∨ ¬VũKhí(y) ∨ ¬Bán(x,y,z) ∨ ¬ThùĐịch(z) ∨ TộiPhạm(x).

Mọi câu logic bậc nhất đều có thể chuyển về dạng câu CNF tương đương logic. Câu CNF sẽ không thỏa mãn chỉ khi câu ban đầu không thỏa mãn, do đó ta có cơ sở để chứng minh phản chứng trên câu CNF.

Thủ tục chuyển thành CNF rất giống với trường hợp mệnh đề. Điểm khác biệt chủ yếu là cần phải loại bỏ lượng từ tồn tại. Ví dụ với câu: "Những người yêu tất cả thú vật đều được yêu bởi ai đó" hay:

$$\forall x\ [\forall y ThúVật(y) \Rightarrow Yêu(x, y)] \Rightarrow [\exists y\ \text{Yêu}(y, x)]$$

Các bước như sau:

Loại bỏ dấu suy ra:

$$\forall x\ [\neg\forall y\ \neg ThúVật(y) \vee Yêu(x, y)] \vee [\exists y\ \text{Yêu}(y, x)]$$

Chuyển dấu ¬ vào trong: Ngoài các luật thông thường cho liên kết phủ định, ta cần luật cho lượng từ phủ định. Do đó, ta có:

$\neg\forall x\ p$ trở thành: $\exists x \neg p$

$\neg\exists x\ p$ trở thành: $\forall x \neg p$

Câu đang xét trải qua các biến đổi sau:

$$\forall x\ [\exists y \neg (\neg ThúVật(y) \vee Yêu(x, y))] \vee [\exists y\ \text{Yêu}(y, x)].$$

$$\forall x\ [\exists y \neg \neg ThúVật(y) \wedge \neg Yêu(x, y)] \vee [\exists y\ \text{Yêu}(y, x)].$$

$$\forall x\ [\exists y\ ThúVật(y) \wedge \neg Yêu(x, y)] \vee [\exists y\ \text{Yêu}(y, x)].$$

Lưu ý lượng từ phổ quát ($\forall y$) trong tiền đề của câu kéo theo được chuyển thành lượng từ tồn tại. Bây giờ câu đọc thành "Hoặc là có một số thú vật mà x không yêu, hoặc (nếu không phải trường hợp này) ai đó yêu x". Rõ ràng là ý nghĩa của câu ban đầu được bảo toàn.

Chuẩn hóa biến: Với các câu như $(\forall x P(x)) \vee (\exists x Q(x))$ dùng cùng một biến hai lần, đổi tên của một trong hai biến. Điều này tránh nhầm lẫn khi loại bỏ lượng từ. Do đó, ta có:

$$\forall x\ [\exists y ThúVật(y) \wedge \neg Yêu(x, y)] \vee [\exists z\ \text{Yêu}(z, x)].$$

Skolem hóa: Skolem hóa là quá trình loại bỏ lượng từ tồn tại. Trong trường hợp đơn giản, nó giống như luật Thể hiện Tồn tại trong phần 6.I: chuyển $\exists x\ P(x)$ thành $P(A)$ với A là hằng mới. Tuy nhiên, nếu áp dụng luật này vào câu ví dụ, ta được:

$$\forall x\ [ThúVật(A) \wedge \neg Yêu(x, A)] \vee Yêu(B, x)$$

có nghĩa sai hoàn toàn: *mọi người hoặc là không yêu một động vật A cụ thể hoặc được yêu bởi một thực thể B nào đó.* Trong thực tế, câu gốc cho phép mỗi người không yêu một động vật khác nhau hoặc được yêu bởi một người khác nhau. Như vậy, ta muốn các thực thể Skolem phụ thuộc vào x:

$$\forall x\ [ThúVật(F(x)) \wedge \neg Yêu(x, F(x))] \vee Yêu(G(x), x)$$

F và G ở đây là các hàm Skolem. Qui tắc tổng quát là mọi đối số của hàm Skolem đều là biến lượng từ phổ quát có tầm vực nằm trong vùng mà lượng từ tồn tại xuất hiện. Cũng như với Thể hiện Tồn tại, câu Skolem hóa là thỏa mãn khi câu ban đầu là thỏa mãn.

Loại bỏ lượng từ phổ quát: đến đây, mọi biến còn lại đều phải là biến lượng từ phổ quát. Hơn nữa, câu tương đương với câu có mọi lượng từ phổ quát đẩy về bên trái. Do đó ta có thể bỏ các lượng từ phổ quát:

$$[ThúVật(F(x)) \wedge \neg Yêu(x, F(x))] \vee Yêu(G(x), x).$$

Phân phối $\vee$ trên $\wedge$:

$$[ThúVật(F(x)) \vee Yêu(G(x), x)] \wedge [\neg Yêu(x, F(x))] \vee Yêu(G(x), x)].$$

Bước này có thể cần phải khai triển các phản hội và giao lồng nhau.

Bây giờ câu đã ở dạng CNF và chứa hai mệnh đề. Câu còn có thêm phần giải thích rằng hàm Skolem $F(x)$ ý chỉ động vật có thể không được yêu bởi x và $G(x)$ chỉ ai đó có thể yêu x.

V.2 Luật suy diễn hợp giải

Luật hợp giải cho các mệnh đề bậc nhất chỉ đơn giản là một phiên bản nâng cao của luật hợp giải mệnh đề. Hai mệnh đề, giả sử đã được chuẩn hóa sao cho không có biến nào chung, có thể hợp giải được nếu chúng chứa từ đối ngẫu. Hai từ mệnh đề là đối ngẫu nếu từ này là bù

của từ kia, hai từ bậc nhất là đối ngẫu khi từ này đồng nhất được với phủ định của từ kia. Như vậy ta có:

$$\frac{l_1 \vee ... \vee l_k, m_1 \vee ... \vee m_n}{SUBST(\theta, l_1 \vee ... \vee l_{i-1} \vee l_{i+1} \vee ... \vee l_k \vee m_1 \vee ... \vee m_{j-1} \vee m_{j+1} \vee ... \vee m_n}$$

với ĐồngNhất(l_i, $\neg m_j$) = θ. Ví dụ, ta có thể giải hai mệnh đề:

$[Thú Vật(F(x)) \vee Yêu(G(x), x)]$ và $[\neg Yêu(u, v) \vee \neg Giết(u, v)]$

bằng cách loại đi hai từ: $Yêu(G(x), x)$ và $\neg Yêu(u, v)$ với hàm đồng nhất: $\theta = \{u/G(x), v/x\}$, tạo ra mệnh đề:

$[Thú Vật(F(x)) \vee \neg Giết(G(x), x)]$

Luật vừa trình bày ở trên là luật hợp giải nhị phân vì nó chỉ hợp giải hai từ. Bản thân luật hợp giải nhị phân không thể đưa đến một thủ tục suy diễn đầy đủ. Luật hợp giải đầy đủ giải các tập con từ đồng nhất được trong từng mệnh đề. Một hướng tiếp cận khác là mở rộng phép phân tích phần tử - loại bỏ các từ dư thừa - đối với với trường hợp bậc nhất. Phân tích phần tử mệnh đề giảm hai từ thành một nếu chúng *giống hệt nhau*; phân tích phần tử bậc nhất giảm hai từ thành một nếu chúng *đồng nhất được*. Hàm đồng nhất phải được áp dụng cho toàn bộ mệnh đề. Tổ hợp của hợp giải nhị phân và phân tích phần tử là đầy đủ.

Ví dụ chứng minh

Hợp giải chứng minh rằng KB |= α bằng cách chứng minh KB $\wedge\neg\alpha$ không thỏa mãn, tức là: suy ra mệnh đề rỗng. Xét hai ví dụ chứng minh sau. Ví dụ đầu tiên là bài toán tội phạm ở phần 6.III. Các câu CNF là:

$\neg NgườiMỹ(x) \vee \neg VũKhí(y) \vee \neg Bán(x,y,z) \vee \neg ThùĐịch(z) \vee TộiPhạm(x)$.

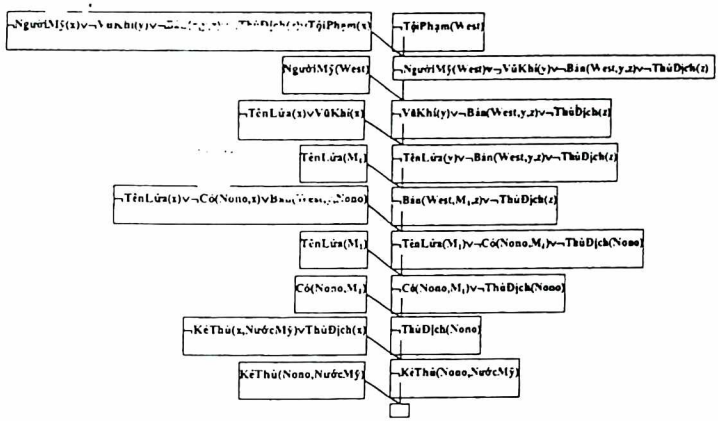
$\neg TênLửa(x) \vee \neg Có(Nono, x) \vee Bán(West, y, Nono)$.

$\neg KẻThù(x, NgườiMỹ) \vee ThùĐịch(x)$

$\neg TênLửa(x) \vee VũKhí(x)$.

$Có(Nono, M_1)$. $TênLửa(M_1)$.

NgườiMỹ(West). *KẻThù(Nono, NướcMỹ).*



Hình 6.8 - *Quá trình chứng minh hợp giải rằng West là tội phạm.*

Phủ định của câu hỏi đích $\neg TộiPhạm(West)$ cũng được thêm vào. Quá trình chứng minh hợp giải được thể hiện trong Hình 6.10. Hãy chú ý đến cấu trúc: một trục duy nhất bắt đầu từ mệnh đề câu hỏi đích, hợp giải với các mệnh đề trong cơ sở tri thức cho đến khi phát sinh mệnh đề rỗng. Đây là đặc điểm của hợp giải trên các cơ sở tri thức mệnh đề Horn. Thực tế là các mệnh đề dọc theo trục tương ứng hoàn toàn với các giá trị liên tiếp của biến câu hỏi đích của thuật toán suy diễn lùi trong Hình 6.5. Điều này là do chúng ta luôn chọn hợp giải mệnh đề có từ dương với từ trái nhất của mệnh đề hiện tại trên xương sống, giống y những gì xảy ra trong suy diễn lùi. Như vậy, suy diễn lùi thực chất là trường hợp đặc biệt của hợp giải với chiến thuật quản lý cụ thể để quyết định thực hiện hợp giải nào kế tiếp.

Ví dụ thứ hai sử dụng Skolem hóa và có liên quan đến các mệnh đề không xác định. Quá trình chứng minh tương đối phức tạp hơn. Bài toán này phát biểu như sau:

Người nào yêu mọi con vật đều được yêu bởi ai đó.

Người nào giết một con vật sẽ không được ai yêu.

Tuấn yêu mọi con vật.

Tuấn hoặc Tình giết con mèo có tên là Mướp.

Có phải Tình đã giết con mèo hay không?

Đầu tiên, ta biểu diễn các câu gốc, một số tri thức nền tảng và phủ định câu hỏi đích G bằng logic bậc nhất:

A. $\forall x\ [\forall y ThúVật(y) \Rightarrow Yêu(x, y)] \Rightarrow [\exists y Yêu(y, x)]$

B. $\forall x\ [\exists y ThúVật(y) \wedge Giết(x, y)] \Rightarrow [\forall z \neg Yêu(z, x)]$

C. $\forall x\ ThúVật(x) \Rightarrow Yêu(Tuấn, x)$

D. $Giết(Tuấn, Mướp) \vee Giết(Tình, Mướp)$

E. $Mèo(Mướp)$

F. $\forall x\ Mèo(x) \Rightarrow ThúVật(x)$

G. $\neg Giết(Tình, Mướp)$

Bây giờ chuyển từng câu trên sang CNF:

A1. $ThúVật(\mathrm{F}(x)) \vee Yêu(G(x), x)$

A2. $\neg Yêu(x, F(x)) \vee Yêu(G(x), x)$

B. $\neg ThúVật(y) \vee \neg Giết(x, y) \vee \neg Yêu(z, x)$

C. $\neg ThúVật(x) \vee Yêu(Tuấn, x)$

D. $Giết(Tuấn, Mướp) \vee Giết(Tình, Mướp)$

E. $Mèo(Mướp)$

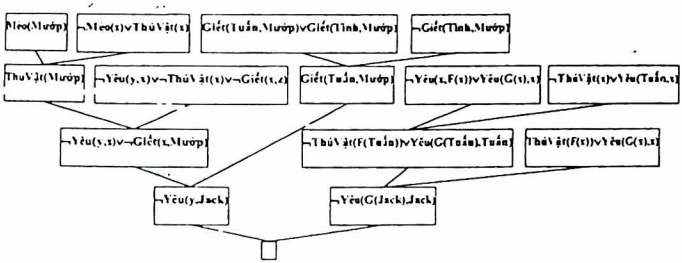
F. $\neg Mèo(x) \vee ThúVật(x)$

G. $\neg Giết(Tình, Mướp)$

Quá trình hợp giải chứng minh Tình giết con mèo được thể hiện trong Hình 6.9. Nội dung được diễn giải như sau:

> Giả sử Tình không giết Mướp. Ta biết Tuấn hoặc Tình đã làm, như vậy nhất định là Tuấn giết. Mướp là con mèo và mèo là động vật, vậy Mướp là một con vật. Vì người nào giết một con vật sẽ không được ai yêu, ta biết được không ai yêu Tuấn. Mặt khác,

Tuấn yêu mọi con vật, vậy ai đó yêu anh ta; như thế ta có mẫu thuẫn. Vậy, Tình đã giết con mèo:



Hình 6.9 - *Chứng minh hợp giải rằng Tình giết con mèo.*

Phép chứng minh trả lời câu hỏi "Có phải Tình đã giết con mèo hay không?", nhưng thường thì ta cần đặt những câu hỏi tổng quát hơn, chẳng hạn như "Ai giết con mèo?" Hợp giải có thể làm điều đó, nhưng cần phải làm thêm một chút việc để đạt được lời giải. Câu hỏi đích là: $\exists w$ *Giết*(*w*, *Mướp*), khi phủ định trở thành: ¬*Giết*(*w*, *Mướp*) ở dạng CNF. Lặp lại quá trình chứng minh trong Hình 6.9 với câu hỏi đích phủ định mới, ta có cây chứng minh tương tự như với phép thế: {*w*/*Tình*} tại một trong các bước. Vậy, trong trường hợp này, việc tìm ra kẻ giết con mèo chỉ là vấn đề theo dõi phép kết cho các biến truy vấn trong khi chứng minh.

Thật không may là hợp giải có thể tạo ra những **chứng minh không suy diễn được** cho câu hỏi đích đến đã có. Ví dụ, ¬*Giết*(*w*,*Mướp*) hợp giải với *Giết*(*Tuấn*, *Mướp*) ∨ *Giết*(*Tình*, *Mướp*) cho ra: Giết(*Tuấn*, *Mướp*), đem hợp giải một lần nữa với: ¬*Giết*(*w*, *Mướp*) cho ra mệnh đề rỗng. Ta thấy *w* có hai phép kết khác nhau trong chứng minh này, hợp giải cho ta biết rằng, đúng là có ai đó đã giết Mướp – hoặc là Tuấn hoặc là Tình. Một giải pháp là giới hạn số bước hợp giải cho phép để biến truy vấn chỉ được kết một lần trong một lượt chứng minh và ta cần có khả năng truy ngược các phép kết có thể có. Một giải pháp khác là thêm một **từ trả lời** vào phủ định câu hỏi đích, ¬*Giết*(*w*, *Mướp*) ∨ *TrảLời*(*w*). Bây giờ, quá trình hợp giải sẽ có câu trả lời khi mệnh đề được phát sinh chỉ chứa một từ trả lời đơn lẻ. Đối với chứng

minh trong Hình 6.9, đó là *TrảLời*(*Tình*). Chứng minh không suy diễn được tạo ra mệnh đề *TrảLời*(*Tình*) ∨ *TrảLời*(*Tuấn*), không thiết lập câu trả lời.

V.3 Các chiến lược hợp giải

Chúng ta biết rằng áp dụng nhiều lần luật suy diễn hợp giải cuối cùng sẽ tìm ra được phép chứng minh nếu như nó thật sự tồn tại. Trong phần này, chúng ta sẽ khảo sát các chiến lược giúp tìm ra chứng minh một cách hiệu quả.

Ưu tiên đơn vị

Chiến lược này ưu tiên hợp giải khi một trong các câu là từ đơn (còn gọi là **mệnh đề đơn vị**). Ý tưởng của chiến lược này là cố gắng tạo ra một mệnh đề rỗng, nên nếu chọn những suy diễn tạo ra những mệnh đề ngắn hơn sẽ tốt hơn. Hợp giải câu đơn vị (chẳng hạn P) với câu khác bất kì (chẳng hạn như $\neg P \vee \neg Q \vee R$) luôn tạo ra một mệnh đề ngắn hơn mệnh đề khác (trong trường hợp này, $\neg Q \vee R$). Khi chiến thuật ưu tiên đơn vị được thực hiện lần đầu tiên trong suy diễn mệnh đề vào năm 1964, nó đã làm tăng tốc đáng kể. giúp chứng minh các định lý mà trước đây không có chiến thuật này thì không làm được. Bản thân ưu tiên đơn vị không làm chuyện đó nhưng nó làm giảm hệ số nhánh cho bài toán cỡ trung bình và giúp cho bài toán giải được bằng hợp giải. Đây là một *heuristic* hữu dụng có thể kết hợp với các chiến thuật khác.

Hợp giải đơn vị là một dạng giới hạn của hợp giải, tại mỗi bước hợp giải phải dùng một mệnh đề đơn vị. Hợp giải đơn vị về mặt tổng quát là không đầy đủ nhưng nó đầy đủ đối với cơ sở tri thức Horn. Chứng minh hợp giải đơn vị trên cơ sở tri thức Horn tương tự như suy diễn tiến.

Tập hỗ trợ

Ưu tiên chọn những hợp giải rõ ràng là có ích nhưng về mặt tổng quát, sẽ hiệu quả hơn nếu ta cũng bỏ được một số hợp giải tiềm năng. Chiến thuật tập hỗ trợ làm được điều này. Nó bắt đầu bằng việc nhận diện một tập con gồm các câu, gọi là tập hỗ trợ. Mọi phép hợp giải sử dụng một câu từ tập hỗ trợ với một câu khác và thêm kết quả hợp giải vào

tập hỗ trợ. Nếu tập hỗ trợ nhỏ so với toàn bộ cơ sở tri thức, không gian tìm kiếm sẽ giảm đáng kể.

Ta cần phải cẩn thận với phương pháp này, bởi vì lựa tập hỗ trợ không tốt sẽ làm cho thuật toán không đầy đủ. Tuy nhiên, nếu chọn tập hỗ trợ S sao cho các câu còn lại cùng thỏa mãn được, thì hợp giải tập hỗ trợ sẽ đầy đủ. Một cách phổ biến là dùng phủ định câu hỏi đích làm tập hỗ trợ, giả sử rằng cơ sở tri thức gốc là nhất quán (nếu nó không nhất quán thì việc truy vấn từ đây trở nên vô ích). Chiến thuật tập hỗ trợ còn có thêm một lợi điểm về việc phát sinh cây chứng minh, con người dễ dàng hiểu cây này bởi vì nó hướng tới câu hỏi đích.

Hợp giải đầu vào

Trong chiến thuật **hợp giải đầu vào**, mọi bước hợp giải kết hợp một trong những câu đầu vào (từ KB hoặc truy vấn) với một số câu khác. Chứng minh trong Hình 6.10 chỉ sử dụng hợp giải đầu vào, có hình dạng như một xương sống, các câu đơn kết hợp lên xương sống. Rõ ràng là không gian tìm kiếm của cây chứng minh dạng này nhỏ hơn so với không gian của mọi đồ thị chứng minh. Trong cơ sở tri thức Horn, tam đoạn luận là một dạng chiến thuật hợp giải đầu vào vì nó kết hợp câu suy dẫn từ KB ban đầu với những câu khác. Như vậy, không có gì ngạc nhiên khi hợp giải đầu vào là đầy đủ với những cơ sở tri thức ở dạng Horn, nhưng không đầy đủ trong trường hợp tổng quát. Chiến thuật **hợp giải tuyến tính** là một sự tổng quát hóa tương đối cho phép P và Q hợp giải nếu P thuộc KB ban đầu hoặc P là tổ tiên của Q trong cây chứng minh. Hợp giải tuyến tính là đầy đủ.

Phương pháp gộp

Phương pháp gộp bỏ đi mọi câu mà được gộp lại (tức là cụ thể hơn) bởi câu đã có trong KB. Ví dụ, nếu $P(x)$ trong KB thì thêm $P(A)$ vào sẽ không có ý nghĩa gì, và ngay cả thêm $P(A) \vee Q(B)$ cũng ít ý nghĩa. Hợp giải phụ giữ KB nhỏ và vì thế giúp cho không gian tìm kiếm nhỏ.

VI. KẾT CHƯƠNG

Chương này đưa ra phân tích về suy diễn logic đối với logic bậc nhất và một số thuật toán thực hiện

- Cách tiếp cận đầu tiên là sử dụng luật suy diễn với các lượng từ được cụ thể hoá để mệnh đề hoá bài toán suy diễn. Thông thường cách tiếp cận này rất chậm.
- Việc sử dụng của đồng nhất để xác định các phép thế thích hợp cho các biến loại bỏ bước cụ thể hoá trong chứng minh bậc nhất, làm cho quá trình trở nên hiệu quả hơn nhiều.
- Một phiên bản nâng cấp của tam đoạn luận sử dụng phép đồng nhất, để cung cấp một luật suy diễn tự nhiên mà mạnh mẽ tên là tam đoạn luận tổng quát. Các thuật toán suy diễn tiến và suy diễn lùi áp dụng luật này để xử lý các mệnh đề xác định.
- Tam đoạn luận tổng quát là đầy đủ đối với các mệnh đề xác định, mặc dù bài toán suy dẫn là bán quyết định. Đối với các chương trình Datalog bao gồm các mệnh đề xác định không có hàm, việc suy dẫn là quyết định được.
- Suy diễn tiến được dùng trong các cơ sở dữ liệu suy diễn trong đó nó có thể được kết hợp với các toán tử cơ sở dữ liệu quan hệ. Suy diễn tiến là đầy đủ đối với các chương trình Datalog và chạy trong thời gian tuyến tính.
- Suy diễn lùi được dùng trong các hệ thống lập trình logic như Prolog.
- Luật suy diễn hợp giải tổng quát cung cấp một hệ thống chứng minh đầy đủ cho logic bậc nhất, sử dụng cơ sở tri thức ở dạng hội chuẩn.

VII.THUẬT NGỮ TIẾNG ANH

Sau đây là những thuật ngữ tiếng Anh liên quan đến những khái niệm đã dùng ở chương này:

answer literal: từ trả lời

existential instantiation: thể hiện tồn tại

first-order definite clause: mệnh đề xác định bậc nhất

first-order literal: từ bậc nhất
fixed point: điểm cố định
Generalized Modus Ponens: tam đoạn luận tổng quát
ground term: biểu thức cơ sở
input resoloution: hợp giải đầu vào
logic programming: lập trình logic
most general unifier (MGU): hàm đồng nhất tổng quát nhất
negation as failure: phủ định khi thất bại
nonconstructive proof: chứng minh không suy diễn được
occur check: kiểm tra xuất hiện
propositionalization: mệnh đề hoá
prototype: nguyên mẫu
semidecidable: bán quyết định
set of support: tập hỗ trợ
Skolem constant, skolem function: hằng Skolem
Skolemization: Skolem hoá
standardizing apart: phân chia chuẩn hoá
substitution: phép thế
subsumption: phương pháp gộp
unification: đồng nhất
unifier: hàm đồng nhất
unit clause: mệnh đề đơn vị
unit preference: ưu tiên đơn vị
unit resolution: hợp giải đơn vị
Universal Instantiation: thể hiện phổ quát

Chương 7
SUY DIỄN KHÔNG CHẮC CHẮN

I. GIỚI THIỆU VỀ SỰ KHÔNG CHẮC CHẮN

Một trong những đặc điểm chung của thông tin hiện có của các chuyên gia con người là tính không hoàn hảo. Thông tin có thể không đầy đủ, không nhất quán, không chắc chắn, hay tất cả ba thứ. Nói cách khác, thông tin thường không phù hợp để giải quyết một vấn đề. Tuy nhiên, một chuyên gia có thể giải quyết những khuyết điểm này và có thể thường dùng để đưa ra các phán đoán và quyết định đúng đắn. Các hệ thống chuyên gia cũng phải có thể giải quyết sự không chắc chắn và rút ra các kết luận có giá trị.

I.1 Sự không chắc chắn là gì?

Sự không chắc chắn có thể được định nghĩa là sự thiếu tri thức chắc chắn cho phép chúng ta đạt được kết luận đáng tin cậy hoàn hảo. Logic truyền thống chỉ cho phép suy diễn chính xác. Nó giả định rằng tri thức hoàn hảo luôn tồn tại và luật bài trung có thể luôn luôn được áp dụng:

IF A là *true*
THEN A không là *false*

và

IF B là *false*
THEN B không là *true*

Thật không may hầu hết các bài toán trong thế giới thực mà các hệ chuyên gia có thể được sử dụng lại không cung cấp tri thức rõ gọn như thế. Ứng dụng chẩn đoán – cho y khoa, sửa chữa xe máy... – là một ứng dụng mà tri thức hầu như không chắc chắn. Hãy thử viết các luật

cho chẩn đoán nha khoa bằng logic bậc nhất và ta có thể thấy rằng các tiếp cận logic bị thất bại. Xem xét luật sau:

$\forall p$ *Triệu_chứng(p, Đau răng)* $\Rightarrow$ *Bệnh(p, Sâu răng).*

Vấn đề là luật trên sai. Không phải tất cả bệnh nhân đau răng đều bị sâu răng; một số họ có bị bệnh về lợi, nhiễm trùng, hoặc một trong số những vấn đề khác:

$\forall p$ *Triệu_chứng(p, Sâu răng)* $\Rightarrow$
Bệnh(p, Đau răng) $\vee$ *Bệnh(p, Viêm lợi)* $\vee$ *Bệnh(p, Nhiễm trùng)*...

Không may là để làm cho luật đúng, cần phải bổ sung một danh sách hầu như không giới hạn các nguyên nhân có thể. Ta có thể thử biến đổi luật trên thành một luật nguyên nhân như sau:

$\forall p$ *Bệnh(p, Sâu răng)* $\Rightarrow$ *Triệu_chứng(p, Đau răng)*

Nhưng luật này cũng không chính xác vì không phải tất cả người bị sâu răng đều đau răng. Cách duy nhất để sửa luật là thực hiện vét cạn logic: bổ sung thêm vế trái giá trị định lượng mà một người bị sâu bao nhiêu sẽ gây nên đau răng. Ngay cả khi đó, đối với mục đích chẩn đoán, ta cũng cần xem xét đến khả năng một bệnh nhân vừa bị đau vừa bị sâu răng nhưng hai cái này không liên quan gì đến nhau.

I.2 Các nguyên nhân gây nên sự không chắc chắn

Sự không chắc chắn trong tri thức có thể có các nguồn gốc sau:

- Chúng ta có thể không biết hết mọi thứ do:
  - **Sự ngu dốt** – chúng ta không biết tất cả mọi thứ hoặc **sự lười biếng** – chúng ta không thể thực hiện tất cả các thí nghiệm để rút ra tri thức đầy đủ cần thiết.
  - Trong nhiều trường hợp, chúng ta không thể đợi để thực hiện các thử nghiệm cần thiết, ví dụ đối với những ca cấp cứu y tế, bệnh nhân có thể chết trước khi các xét nghiệm được hoàn thành.
- Sự nhập nhằng về biểu diễn:

- Con người có thể biểu diễn một sự vật khác nhau bằng nhiều cách khác nhau.
- Thiếu một tiêu chuẩn cần thiết để đánh giá các sự vật. Ví dụ trong câu *Người già cần đeo kính?*, thì cần đánh giá như thế nào gọi là *già*?.

- Sự không chính xác của thông tin: thông tin được ghi nhận và truyền đạt bởi con người hay các thiết bị máy móc. Do đó thông tin cũng chịu sự không chính xác do những sai lệch trong cảm nhận của con người và thiết bị.

Các luật thường là các *heuristic* mà các chuyên gia sử dụng trong các tình huống cho trước. Chúng không nhất thiết phải hoàn hảo!.

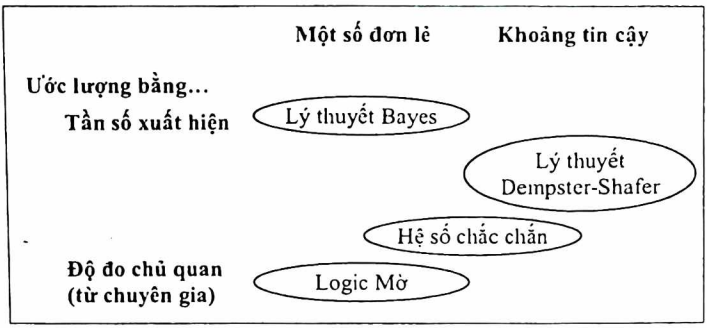
1.3 Các mô hình biểu diễn không chắc chắn

Như đã phân tích, thông tin không chắc xuất hiện rộng rãi trong đời sống hàng ngày. Việc rút ra các kết luận từ thông tin không hoàn hảo và kém chính xác hay dùng các suy diễn không chính xác vẫn được thực hiện. Thực tế cho thấy con người rất thành công trong việc suy diễn này, ví dụ như trong lĩnh vực chẩn đoán y khoa, phát hiện và xử lý sự cố...

Một số mô hình suy diễn không chắc chắn đã được vận dụng xây dựng các hệ chuyên gia trên máy tính. Hình 7.1 cho biết một số mô hình và cách thức biểu diễn sự không chắc chắn của chúng.

Trong các vấn đề không chắc chắn, mỗi tri thức được cung cấp một **mức độ tin cậy**. Tùy thuộc vào việc biểu diễn mức độ tin cậy này mà ta phân chia các cách tiếp cận. Lý thuyết xác suất với cách tiếp cận *Bayes* là công cụ chính được sử dụng để giải quyết những bài toán không chắc chắn, sử dụng giá trị xác suất để biểu diễn độ tin cậy. Đây là một giá trị số học nằm trong khoảng từ 0 đến 1 và được ước lượng khách quan từ dữ liệu của bài toán. Lý thuyết Dempster-Shafer là trường hợp tổng quát của lý thuyết xác suất, biểu diễn độ tin cậy bằng một khoảng giá trị. Cách tiếp cận hệ số chắc chắn cũng biểu diễn mức độ tin cậy bằng hai giá trị cung cấp bởi các ước đoán chủ quan của chuyên gia. Sau đó phương pháp này kết hợp các giá trị tin cậy vào một hệ số chắc chắn và sử dụng nó để suy diễn. Logic mờ biểu diễn

mức độ tin cậy với những từ mô tả ngữ nghĩa theo cách nói tự nhiên của con người.



Hình 7.1 - *Lược đồ mô tả một số mô hình biểu diễn tri thức không chắc chắn. Các mô hình được phân loại theo cách ước lượng và biểu diễn giá trị không chắc chắn.*

Phần tiếp theo sẽ trình bày về hai hệ thống phổ biến: *suy diễn dựa trên tiếp cận Bayes* và *suy diễn dựa trên hệ số chắc chắn*. Hệ thống thứ nhất dựa trên lý thuyết xác suất thống kê, vốn được ứng dụng rất rộng rãi trong nhiều lĩnh vực của khoa học máy tính như: học máy, nhận dạng, khai thác dữ liệu, truy vấn thông tin...Hệ thống thứ hai là một cải biên từ cái đầu, trong đó đưa vào các tri thức do các chuyên gia cung cấp để cải thiện khả năng suy diễn.

II. LÝ THUYẾT XÁC SUẤT CỦA *BAYES*

Suy luận *Bayes* dựa trên lý thuyết xác suất hình thức và được sử dụng rộng rãi trong một số lĩnh vực nghiên cứu hiện nay, bao gồm cả nhận dạng và phân loại. Trước khi đi vào lý thuyết về *Bayes*, ta hãy cùng nhau xem lại một số khái niệm cơ bản của lý thuyết xác suất.

II.1 Mệnh đề

Độ tin cậy thường được áp dụng lên các mệnh đề hay sự kiện. Đến thời điểm này, ta đã được khảo sát hai loại logic – logic mệnh đề và

logic bậc nhất. Lý thuyết xác suất thường sử dụng một loại ngôn ngữ mang tính biểu cảm nhiều hơn một chút so với logic mệnh đề.

Thành phần cơ bản của ngôn ngữ là các **biến ngẫu nhiên**. Các biến ngẫu nhiên có một miền giá trị mà nó có thể nhận. Ví dụ ta có biến ngẫu nhiên *Sâu răng*, cho biết tình trạng răng khôn ở hàm dưới bên trái của bệnh nhân có bị sâu răng hay không, có thể nhận một trong hai giá trị <*true, false*>. Khi đó, *Sâu răng = false* được dùng để biểu diễn một mệnh đề cho biết rằng răng khôn ở hàm dưới của bệnh nhân là không bị sâu. Một mệnh đề như thế còn được gọi là một sự kiện phản ánh một trạng thái của các đối tượng đang quan tâm.

Tuỳ thuộc vào tính chất của miền trị, một biến ngẫu nhiên có thể được chia làm các loại như sau:

- Biến ngẫu nhiên logic: Nhận một trong hai giá trị *true* hay *false*. Đối với biến ngẫu nhiên logic, ta có thể sử dụng dạng viết tắt của mệnh đề, ví dụ *Sâu răng=true* được viết thành *Sâu răng* và *Sâu răng=false* được viết thành ¬*Sâu răng*.
- Biến ngẫu nhiên rời rạc: Nhận giá trị trong một miền đếm được. Ví dụ, miền giá trị của *Mùa* có thể là {*xuân, hạ, thu, đông*}. Các giá trị trong một miền phải vét cạn và loại trừ lẫn nhau. Biến ngẫu nhiên logic là một trường hợp đặc biệt của loại này. Nếu không gặp sự nhập nhằng nào, ta có thể viết tắt *xuân* cho mệnh đề *Mùa=xuân*.
- Biến ngẫu nhiên liên tục: Nhận giá trị từ các số thực. Ví dụ, mệnh đề X=4.02 cho biết biến ngẫu nhiên *X* có giá trị chính xác là 0.42. Các mệnh đề liên quan đến biến ngẫu nhiên liên tục cũng có thể mang dấu bất đẳng thức.

Các mệnh đề như *Sâu răng = false* và *Đau răng = true* có thể được kết hợp thành các mệnh đề phức với các phép nối logic chuẩn, ví dụ: *Sâu răng = false* ∧ *Đau răng = true*. Như đã giải thích ở trên, mệnh đề trên này cũng có thể biểu diễn dưới dạng: ¬*Sâu răng* ∧ *Đau răng*.

II.2 Sự kiện nguyên tố

Một **sự kiện nguyên tố** là một mô tả đầy đủ về trạng thái của thế giới không chắc chắn đang xem xét. Ví dụ, nếu thế giới đang quan sát chỉ

bao gồm hai biến ngẫu nhiên logic *Sâu răng* và *Đau răng*, thì chỉ có bốn sự kiện nguyên tố khác nhau, ví dụ: một trong số các sự kiện là *Sâu răng* = *false* ∧ *Đau răng* = *true*.

Các sự kiện nguyên tố có một số đặc điểm quan trọng:

- Chúng loại trừ lẫn nhau - chỉ có tối đa một sự kiện được diễn ra. Ví dụ: ¬*Sâu răng* ∧ *Đau răng* và¬*Sâu răng* ∧ ¬*Đau răng* không thể xảy ra đồng thời.
- Tập tất cả sự kiện nguyên tố là vét cạn - có ít nhất một sự kiện phải xảy ra.
- Bất kỳ sự kiện nguyên tố nào cũng kéo theo tính đúng hay sai của mọi mệnh đề dù đơn giản hay phức tạp. Ví dụ, sự kiện nguyên tố *Sâu răng* ∧ ¬*Đau răng* kéo theo *Sâu răng* là đúng và *Sâu răng* ⇒ *Đau răng* là sai.
- Bất kỳ mệnh đề nào cũng tương đương logic với nối rời của tất cả sự kiện nguyên tố suy dẫn được tính đúng của mệnh đề đó. Ví dụ, mệnh đề *Sâu răng* tương đương với nối rời của các sự kiện nguyên tố *Sâu rang* ∧ *Đau răng* và *Sâu răng* ∧ ¬*Đau răng*.

II.3 Xác suất tiên nghiệm

Xác suất của một sự kiện hay mệnh đề là tỷ lệ số lần sự kiện đó xảy ra trên tổng số trường hợp. Xác suất có thể được biểu diễn toán học bằng một chỉ số nằm trong khoảng từ không (chắc chắc không thể) đến một (chắc chắn có thể).

Hãy cùng xem xét các ví dụ kinh điển về tung đồng xu và tung xúc xắc. Nếu tung một đồng xu, có hai sự kiện có thể xảy ra là đồng xu *Ngửa* hay *Xấp* và hai sự kiện trên đều có xác suất xảy ra như nhau. Trong một lần tung, xác suất đồng xu: *Ngửa*, P(*Ngửa*) = 0.5.

Xem xét việc tung một con xúc xắc và xác định xác suất để nhận được mặt số 6 từ một lần tung. Nếu ta ký hiện sự kiện *A* = *xúc xắc* có mặt 6 thì xác suất của A là 1/6 vì trong một lần tung, chỉ có 1 khả năng nhận được mặt 6 và năm khả năng còn lại đều không nhận được mặt 6:

$$P(A) = \frac{1}{1+5} = \frac{1}{6}$$

và xác suất không nhận được mặt 6:

$$P(\neg A) = \frac{5}{1+5} = \frac{5}{6}$$

Những xác suất trên được gọi là **xác suất tiên nghiệm** hay **xác suất không điều kiện**, vì nó cho biết độ tin cậy của một sự kiện khi không có bất kỳ thông tin nào khác.

Đối với ví dụ tung xúc xắc, ta có thể nhận được 1 trong 6 sự kiện ứng với các mặt khác nhau của xúc xắc. Trong trường hợp đó, ta có được một **phân phối xác suất** của việc tung xúc xắc như sau:

$$P(X) = \left\langle \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6} \right\rangle$$

Gọi A và B là hai sự kiện đang quan sát. Xác suất mà hai sự kiện đều

xảy ra được gọi là xác suất kết hợp của A và B, ký hiệu toán học là $P(A \wedge B)$. Khi đó bảng phân phối xác suất của hai sự kiện A và B sẽ chứa xác suất xảy ra của tất cả các sự kiện nguyên tố chứa A và B. Ví dụ, một bảng phân phối xác suất của hai sự kiện *Xúc xắc* và *Đồng xu* sẽ chứa 6×2 giá trị. Bảng này được gọi là **phân phối xác suất kết hợp** của *Xúc xắc* và *Đồng xu*.

II.4 Xác suất hậu nghiệm

Đến bây giờ ta mới quan tâm đến các sự kiện độc lập và loại trừ lẫn nhau. Trong ví dụ tung xúc xắc, hai sự kiện thu được mặt 6 và thu được mặt 1 là loại trừ lẫn nhau, vì ta không thể đồng thời thu được mặt 6 và 1 trong một lần tung. Tuy nhiên, các sự kiện không độc lập

nhau có thể ảnh hưởng đến khả năng của một hay những sự kiện khác. Ví dụ, xét xác suất thu được mặt 6 trong một lần tung biết rằng 1 đã không xuất hiện. Vẫn có năm khả năng không thu được mặt 6 nhưng một khả năng đã được loại trừ vì ta biết rằng 1 không xuất hiện do đó xác suất là:

$$P = \frac{1}{1+(5-1)}$$

Gọi A là một sự kiện và B là sự kiện khác. Giả sử rằng sự kiện A và B không loại trừ lẫn nhau, những xuất hiện có điều kiện dựa trên sự xuất hiện của sự kiện kia. Xác suất sự kiện A xảy ra khi sự kiện B xảy ra được gọi là **xác suất có điều kiện** hay **xác suất hậu nghiệm.** Xác xuất hậu nghiệm được ký hiệu là P(A|B), dấu dọc được đọc là "trên" và toàn bộ biểu thức xác suất được diễn đạt là "Xác suất có điều kiện của sự kiện A khi sự kiện B đã xảy ra". Công thức xác định của xác suất có điều kiện:

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

Tương tự, xác suất có điều kiện của sự kiện B xảy ra khi sự kiện A đã xảy ra là:

$$P(B \mid A) = \frac{P(A \wedge B)}{P(A)}$$

Và do đó:

$$P(A \wedge B) = P(B \mid A) \times P(A)$$

III. SUY DIỄN *BAYES*

Với các khái niệm về xác suất vừa được trình bày ở trên, ta trở về với mục tiêu chính của mình đó là giải quyết việc suy diễn với tri thức không chắc chắn. Trong logic, việc suy diễn được thực hiện với các

mệnh đề E ⇒ H, trong đó E là được gọi là các sự kiện chứng cứ (tiền đề) và H là sự kiện giả thuyết (kết luận). Với sự không chắc chắn, ta biểu diễn mệnh đề trên về dạng luật có dạng như sau:

IF *E* là *true*

THEN *H* là *true* {với xác suất *P*}

Trong loại suy diễn này, ta không chỉ đơn thuần chỉ ra việc giả thuyết *H* có xảy ra hay không khi cho trước các tiền đề *E* mà còn xác định xác suất xảy ra các sự kiện đó là bao nhiêu. Do đó việc suy diễn trong mô hình xác suất tương ứng với việc đi tính các giá trị xác suất có điều kiện trong các luật suy diễn P(*H*|*E*).

III.1 Suy diễn với bảng phân phối kết hợp

Một trong những phương pháp để tính xác suất là thu thập dữ liệu và xây dựng bảng phân phối xác suất cho các sự kiện nguyên tố. Từ bảng phân phối xác suất kết hợp, ta có thể tính được bất cứ giá trị nào.

Với ví dụ chẩn đoán nha khoa, ta xây dựng được bảng phân phối xác suất cho ba sự kiện *Đau*, *Trám* và *Sâu* dựa vào số liệu thu thập được như trong Bảng 7-1.

Bảng 7-1. *Phân phối xác suất kết hợp với chẩn đoán nha khoa.*

	Đau		**¬Đau**	
	Trám	**¬Trám**	**Trám**	**¬Trám**
Sâu	0.108	0.012	0.072	0.008
¬Sâu	0.016	0.064	0.144	0.576

Bảng phân phối xác xuất trên cho phép tính được bất kỳ giá trị xác suất nào của các sự kiện. Ví dụ, có thể tính được xác suất của một bệnh nhân bị sâu răng khi đến khám nha sĩ:

$$P(\text{Sâu}) = 0.108 + 0.012 + 0.072 + 0.008 = 0.2$$

Giá trị xác suất trên được tính bằng cách cộng bốn giá trị ở dòng đầu tiên ứng với bốn sự kiện nguyên tố mà *Sâu* là đúng. Thao tác này được gọi là ***hiện hóa***, hay ***tính tổng ngoài*** bởi vì các biến khác *Sâu* được cộng vào. Xác suất có điều kiện cũng có thể được tính từ bảng

phân phối kết hợp. Ví dụ ta muốn tính xác suất một bệnh nhân không bị sâu răng khi có triệu chứng đau răng như sau:

$$P(\neg \text{Sâu} \mid \text{Đau}) = \frac{P(\neg \text{Sâu} \wedge \text{Đau})}{P(\text{Đau})}$$

$$= \frac{(0.016 + 0.064)}{(0.108 + 0.012 + 0.016 + 0.064)} = 0.4$$

Như trên đã thấy, bảng phân phối xác suất kết hợp cho phép ta tính được bất cứ xác suất hậu nghiệm, hay xác suất suy diễn nào. Tuy nhiên, việc lưu trữ các giá trị xác suất này gặp phải một số vấn đề như sau:

- Kích thước bộ nhớ dùng để lưu trữ bảng phân phối kết hợp, theo lý thuyết là $O(d^n)$ với d là số giá trị trung bình của một thuộc tính và n là số thuộc tính.
- Khi tính xác suất: Tính tổng các giá trị xác suất của các sự kiện nguyên tố, độ phức tạp $O(d^n)$.
- Việc tìm kiếm các mục trong bảng phân phối cũng là một vấn đề khi kích thước bảng lớn.

Do các đặc điểm trên, việc suy diễn dựa trên bảng phân phối xác suất là không thể khi tập hợp sự kiện lớn. Ngay cả trong trường hợp bình thường ta cũng cần thu thập một lượng lớn dữ liệu để lập được bảng phân phối xác suất. Để vượt qua trở ngại trên, ta vận dụng đến luật *Bayes* để tính các xác suất có điều kiện.

III.2 Luật *Bayes*

Quay lại với các công thức xác suất ở phần trên, ta có xác suất kết hợp giữa hai sự kiện A và B là:

$$P(A \wedge B) = P(B \mid A) \times P(A)$$

Xác suất có điều kiện của A khi B xảy ra là:

$$P(A \mid B) = \frac{P(A \wedge B)}{P(B)}$$

Từ đó ta có được:

$$P(A \wedge B) = P(B|A) \times P(A) = P(A|B) \times P(B)$$

suy ra:

$$P(B|A) = \frac{P(B|A) \times P(B)}{P(A)}$$

Công thức trên được gọi là luật *Bayes*, được đặt theo tên nhà toán học người Anh thế kỷ 18 Thomas *Bayes*. Công thức này nói rằng xác suất đúng của giả thuyết B khi quan sát được chứng cứ A, bằng với xác xuất cho rằng chúng ta sẽ quan sát được chứng cứ A nếu giả thuyết B là đúng, nhân với xác suất tiên nghiệm của B, tất cả chia cho xác suất tiên nghiệm của việc quan sát được chứng cứ A.

Ví dụ, trong trường hợp chẩn đoán, cho trước một căn bệnh D và một triệu chứng S, thì xác suất một người có triệu chứng S bị bệnh D được tính theo công thức:

$$P(D|S) = \frac{P(D \wedge S)}{P(S)}$$

với $P(D \wedge S)$ là số người mắc bệnh D và có triệu chứng S và $P(S)$ là số người có triệu chứng S.

Để tính P(D|S) theo công thức trên, thì ta phải thực hiện các cuộc điều tra thống kê trên một phạm vi cộng đồng dân cư rộng lớn mới cho ta được một xác suất chính xác. Công thức *Bayes* đưa ra một cách tính khác cho phép ta tính được P(D|S) từ những kết quả thống kê đã có trước đó hoặc những kết quả thống kê dễ đạt được hơn.

III.3 Ví dụ đơn giản

Xét một chẩn đoán với bệnh viêm màng não, đặt các sự kiện S là bệnh nhân có triệu chứng cứng cổ và M là bệnh nhân bị bệnh viêm màng não.

Các xác suất biết trước:

$P(S|M) = 0.5$.

P(M) = 1/50000.

P(S) = 1/20.

Sử dụng luật *Bayes* suy ra: Khả năng bị bệnh viêm màng não khi thấy bệnh nhân có triệu chứng cứng cổ là:

$$P(M \mid S) = \frac{P(S \mid M)P(M)}{P(S)} = \frac{0.5 \times 1/50000}{1/20} = 0.0002$$

Điểm quan trọng của lý thuyết *Bayes* là các con số ở vế phải của công thức dễ dàng xác định, ít nhất là khi so sánh với vế trái. Như trong ví dụ trên, do số lượng nhỏ hơn, nên việc xác định số bệnh nhân bị cúm kèm theo triệu chứng sốt dễ hơn là xác định số người có triệu chứng sốt bị cúm.

P(S|M) chính là tri thức về nguyên nhân, tức là xác suất hậu nghiệm ở vế phải, dễ dàng có được hơn là tri thức về chẩn đoán, P(M|S), tức là xác suất hậu nghiệm ở vế trái. Vì vậy, ta nói luật *Bayes* cho phép chúng ta sử dụng tri thức về nguyên nhân để suy ra tri thức về chẩn đoán.

III.4 Kết hợp các chứng cứ

Khi một sự kiện xảy ra là kết quả của sự kết hợp hai hay nhiều chứng cứ khác nhau, công thức *Bayes* cũng đưa ra một phương pháp tính hiệu quả. Xét ví dụ chẩn đoán nha khoa, bệnh *Sâu* có thể gây ra bởi triệu chứng *Đau* và có vết *Trám*. Xác suất có điều kiện của chẩn đoán trên là:

$$P(Sâu \mid \text{Đau} \wedge \text{Trám})$$

Dựa vào bảng phân phối kết hợp đầy đủ ta có thể tính được giá trị xác suất trên. Tuy nhiên cách tiếp cận này sẽ dẫn đến một sự bùng nổ dữ liệu khi số biến sự kiện lớn. Công thức *Bayes* tổng quát giúp ta giải quyết được trường hợp này. Gọi $E_1, E_2, ..., E_n$ là n chứng cứ cho giả thuyết H. Luật *Bayes* tổng quát đối với trường hợp này là:

$$P(H|E_1E_2\dots E_n)=\frac{P(E_1E_2\dots E_n\,|\,H)\times P(H)}{P(E_1E_2\dots E_n)}$$

Đến lượt mình, để tính được xác suất $P(E_1E_2\dots E_n\,|\,H)$ cũng cần khai triển một số lớn các hệ số. Tuy nhiên, giả định **độc lập có điều kiện** cho phép ta tính được giá trị xác suất trên một cách dễ dàng hơn. Hai sự kiện A và B được gọi là độc lập có điều kiện khi xảy ra sự kiện D khi:

$$P(AB|D)=P(A\,|\,D)\times P(B\,|\,D)$$

Và do đó công thức tính xác suất giả thuyết H dựa trên các chứng cứ $E_1, E_2,\dots, E_n$ được viết thành:

$$P(H|E_1E_2\dots E_n)=\frac{\prod_{i=1}^{n}P(E_i\,|\,H)\times P(H)}{\prod_{i=1}^{n}P(E_i)}$$

các giá trị $P(E_i|H)$ và $P(E_i)$ có thể được thống kê và tính dễ dàng trên các tập dữ liệu cho cơ sở tri thức.

IV. LÝ THUYẾT HỆ SỐ CHẮC CHẮN VÀ SUY DIỄN DỰA TRÊN CHỨNG CỨ

IV.1 Hệ số chắc chắn

Hệ số chắc chắn là một thay thế phổ biến của suy diễn *Bayes*. Nguyên lý cơ bản của lý thuyết này được giới thiệu trong MYCIN, một hệ chuyên gia chẩn đoán và điều trị nhiễm trùng máu và viêm màng não. Những nhà phát triển của MYCIN thấy rằng các chuyên gia y tế biểu diễn mức độ tin cậy của họ theo các từ ngữ không có tính logic hay toán học nào. Ngoài ra, dữ liệu thống kê tin cậy về lĩnh vực bài toán này cũng không có. Do đó, nhóm MYCIN không thể dùng cách tiếp cận xác suất. Thay vào đó họ giới thiệu một ***hệ số chắc chắn (cf)***, một số đo độ tin cậy của chuyên gia. Giá trị lớn nhất của hệ số chắc chắn là +1.0 (hoàn toàn đúng) và nhỏ nhất là –1.0 (hoàn toàn sai). Một giá trị dương biểu diễn mức độ tin cậy và giá trị âm biểu diễn mức độ không tin cậy.

Trong các hệ chuyên gia dùng hệ số chắc chắn, cơ sở tri thức bao gồm một tập các luật có cú pháp sau:

IF <chứng cứ>

THEN <giả thuyết> {cf}

Trong đó *cf* biểu diễn mức độ tin cậy của giả thuyết H khi chứng cứ E xảy ra.

Hệ số chắc chắn dựa trên hai hàm: độ đo tin cậy *MB(H,E)* và độ đo không tin cậy *MD(H,E)*. Hai hàm này chỉ ra mức độ tin cậy của giả thuyết H sẽ tăng lên bao nhiêu nếu chứng cứ E được quan sát thấy và mức độ không tin cậy của giả thuyết H sẽ tăng lên bao nhiêu nếu chứng cứ E được quan sát thấy.

Độ đo tin cậy và không tin cậy có thể được định nghĩa theo xác suất tiên nghiệm và xác suất có điều kiện như sau:

$$MB(H,E)=\begin{cases}1 & \text{khi } p(H)=1\\ \dfrac{\max\left[p(H|E),p(H)\right]-p(H)}{\max[1,0]-p(H)} & \text{TH khác}\end{cases}$$

$$MD(H,E)=\begin{cases}1 & \text{khi } p(H)=0\\ \dfrac{\max\left[p(H|E),p(H)\right]-p(H)}{\max[1,0]-p(H)} & \text{TH khác}\end{cases}$$

trong đó:

p(H) là xác suất tiên nghiệm của giả thuyết H là đúng.

p(H|E) là xác suất giả thuyết H đúng khi chứng cứ E đúng.

Hai giá trị *MB(H, E)* và *MD(H, E)* nằm trong khoảng 0 và 1. Độ lớn của độ tin cậy hay không tin cậy của giả thuyết *H* phụ thuộc vào loại chứng cứ *E* quan sát được. Một số sự kiện có thể làm tăng độ lớn của độ tin cậy, nhưng một số khác lại làm tăng độ lớn của độ không tin cậy.

IV.2 Các phép toán

Để kết hợp độ lớn của độ tin cậy và độ không tin cậy trong một giả thuyết, công thức sau được sử dụng:

$$cf = \frac{MB(H,E) - MD(H,E)}{1 - \min[MB(H,E), MD(H,E)]}$$

Và qua đó *cf*, *cf* thuộc đoạn -1 đến +1, sẽ chỉ ra độ tin cậy tổng cộng của một giả thuyết H.

Cách tiếp cận MYCIN có thể được mô tả qua ví dụ sau. Xét một luật đơn giản:

IF A là X

THEN B là Y

Thông thường, một chuyên gia có thể không chắc chắn tuyệt đối luật trên sẽ thỏa. Cũng giả sử rằng sau một số lần quan sát, ngay cả khi mệnh đề IF của luật thỏa và đối tượng A nhận giá trị X, đối tượng B có thể nhận giá trị Z khác. Chuyên gia thường kết hợp một hệ số chắc chắn với từng giá trị có thể có của B khi A có giá trị X. Do đó, luật sẽ có dạng:

IF A là X

THEN B là Y {*cf* 0.7};

B là Z {*cf* 0.2}

Điều đó có nghĩa rằng, khi A nhận giá trị X, B sẽ là Y với 70 phần trăm và Z trong 20 phần trăm thời gian. Mười phần trăm thời gian còn lại có thể giá trị khác nữa vì chuyên gia vẫn còn bảo lưu khả năng đối tượng B có thể không chỉ nhận hai giá trị đã biết X và Y mà còn có giá trị khác chưa quan sát được.

Hệ số chắc chắc được gán bởi một luật sau đó được lan truyền qua chuỗi suy diễn. Sự lan truyền của hệ số chắc chắn bao gồm việc thiết lập độ chắc chắn thực của kết quả luật khi chứng cứ trong tiền đề luật

là không chắc chắn. Độ chắc chắn thực đối với một luật có một tiền đề, *cf(H,E)*, có thể được tính dễ dàng bằng cách nhân hệ số chắc chắn của tiền đề, *cf(E)*, với hệ số chắc chắn luật, *cf*:

$$cf(H,E) = cf(E) \times cf$$

Ví dụ:

IF trời trong

THEN dự đoán là trời nắng {*cf* 0.8}

và độ chắc chắn hiện tại của *trời trong* là 0.5, thì:

$$cf(H,E) = 0.5 \times 0.8 = 0.4$$

IV.3 Suy diễn dựa trên chứng cứ

Đối với các luật nối liền như:

IF <chứng cứ E_1>

AND <chứng cứ E_2>

...

AND <chứng cứ E_n>

THEN <giả thuyết*H*> {*cf*}

độ chắc chắn lưới của kết luận, giả thuyết H, được tính như sau:

$$cf(H, E_1 \cap E_2 \cap \ldots \cap E_n) = \min[cf(E_1), cf(E_2), \ldots, cf(E_n)] \times cf$$

Ví dụ:

IF trời trong

AND dự đoán là trời nắng

THEN hành động là 'đeo kính râm' {*cf* 0.8}

và độ chắc chắn của *trời trong* là 0.9 và độ chắc chắn của *dự đoán là trời nắng* là 0.7 thì:

$$cf(H, E_1 \cap E_2) = \min[0.9, 0.7] \times 0.8 = 0.7 \times 0.8 = 0.56$$

Đối với luật nối rời như:

IF <chứng cứ E_1>

OR <chứng cứ E_2>

...

OR <chứng cứ E_n>

THEN <giả thuyếtH> {cf}

độ chắc chắn của giả thuyết H được tính như sau:

$$cf(H, E_1 \cup E_2 \cup \ldots \cup E_n) = \max[cf(E_1), cf(E_2), \ldots, cf(E_n)] \times cf$$

Ví dụ:

IF trời âm u

OR dự đoán là trời mưa

THEN hành động là 'đem theo dù' {cf 0.9}.

và độ chắc chắn của *trời âm u* là 0.6 và độ chắc chắn của *dự đoán là trời mưa* là 0.8 thì:

$$cf(H, E_1 \cup E_2) = \max[0.6, 0.8] \times 0.9 = 0.8 \times 0.9 = 0.72$$

Khi việc thực thi hai luật trở lên có kết quả là cùng một kết luận, hệ số chắc chắn đơn lẻ của những luật này phải được trộn lại để đưa đến hệ số chắc chắn kết hợp cho một giả thuyết. Giả sử cơ sở tri thức bao gồm các luật sau:

Luật 1: IF A là X

THEN C là Z {cf 0.8}

Luật 2: IF B là Y

THEN C là Z {cf 0.6}

Nếu cả Luật 1 và Luật 2 được thực thi thì độ chắc chắn của đối tượng C được gán là bao nhiêu?. Cảm giác chung của ta là: nếu có hai chứng cứ (A là X và B là Y) từ hai nguồn khác nhau (Luật 1 và Luật 2) cùng

hỗ trợ một giả thuyết (C là Z), thì độ tin cậy trong giả thuyết sẽ tăng lên và trở nên vững chắc hơn so với khi chỉ thu được một chứng cứ.

Độ chắc chắn kết hợp được tính toán theo công thức:

$$cf(cf_1, cf_2) = \begin{cases} cf_1 + cf_2 \times (1 - cf_1) & khi\ cf_1 > 0\ và\ cf_2 > 0 \\ \dfrac{cf_1 + cf_2}{1 - \min[|cf_1|, |cf_2|]} & khi\ cf_1 < 0\ hay\ cf_2 < 0 \\ cf_1 + cf_2 \times (1 + cf_1) & khi\ cf_1 < 0\ và\ cf_2 < 0 \end{cases}$$

Trong đó:

cf_1 là độ tin cậy của giả thuyết H đạt được bởi Luật 1;

cf_2 là độ tin cậy của giả thuyết H đạt được bởi Luật 2;

$|cf_1|$ và $|cf_1|$ là trị tuyệt đối của cf_1 và cf_2.

Do đó nếu ta giả sử rằng:

$$cf(E_1) = cf(E_2) = 1.0$$

thì ta tính được:

$$cf_1(H, E_1) = cf(E_1) \times cf_1 = 1.0 \times 0.8 = 0.8$$

$$cf_2(H, E_2) = cf(E_2) \times cf_2 = 1.0 \times 0.6 = 0.6$$

và

$$cf(cf_1, cf_2) = cf_1(H, E_1) \times cf_2(H, E_2) \times (1 - cf_1(H, E_1)$$

$$= 0.8 + 0.6 \times (1 - 0.8) = 0.92$$

Ví dụ trên trình bày một sự tăng cường của độ tin cậy trong một giả thuyết. Bây giờ hãy xét một trường hợp khi mà các hệ số chắc chắn có dấu trái nhau. Giả sử:

$$cf(E_1) = 1.0 \text{ và } cf(E_2) = -1.0$$

thì:

$$cf_1(H, E_1) = cf(E_1) \times cf_1 = 1.0 \times 0.8 = 0.8$$
$$cf_2(H, E_2) = cf(E_2) \times cf_2 = -1.0 \times 0.6 = -0.6$$

và

$$cf(cf_1, cf_2) = \frac{cf_1(H, E_1) + cf_2(H, E_2)}{1 - \min[|cf_1(H, E_1)|, |cf_2(H, E_2)|]} = \frac{0.8 - 0.6}{1 - \min[0.8, 0.2]} = 0.5$$

Ví dụ trên trình bày trường hợp khi một luật xác nhận một giả thuyết và luật kia phủ định giả thuyết thì hệ số chắc chắn kết hợp, hay độ tin cậy thực là bao nhiêu. Trong trường hợp cả hai chứng cứ đều bị bác bỏ, nghĩa là: $cf(E_1) = cf(E_2) = 1.0$, hệ số chắc chắn kết hợp được tính tương tự ví dụ đầu tiên nhưng có dấu âm.

Lý thuyết hệ số chắc chắn cung cấp một phương pháp thay thế cho suy diễn *Bayes*. Phương thức *heuristic* kết hợp các hệ số chắc chắn khác biệt với phương thức được dùng để kết hợp các xác suất. Lý thuyết chắc chắn không "thuần tuý toán học" mà nó bắt chước quá trình suy nghĩ của các chuyên gia con người.

V. SO SÁNH SUY DIỄN *BAYES* VÀ HỆ SỐ CHẮC CHẮN

Các phần trước đã trình bày hai kỹ thuật thông dụng nhất để xử lý sự không chắc chắn trong các hệ chuyên gia. Phần này sẽ so sánh hai kỹ thuật trên và xác định loại bài toán nào có thể được sử dụng hiệu quả bằng suy diễn *Bayes* hay hệ số chắc chắn.

Lý thuyết xác suất là kỹ thuật lâu đời nhất và có nền tảng vững chắc nhất để xử lý tri thức không chính xác và dữ liệu ngẫu nhiên. Nó hoạt động tốt trong những lĩnh vực như dự đoán và lập kế hoạch, trong đó dữ liệu thường có sẵn và các phát biểu xác suất chính xác có thể được thực hiện.

Một hệ chuyên gia áp dụng kỹ thuật *Bayes*, PROSPECTOR, được phát triển để giúp đỡ những nhà thăm dò địa chất tìm kiếm các mỏ tài nguyên. Hệ thống rất thành công, ví dụ với việc dùng các dữ liệu địa chất, địa lý và địa hóa học, PROSPECTOR dự đoán sự tồn tại của *Molypdden* gần ngọn núi *Tolman* bang *Washington*, Hoa Kỳ. Tuy nhiên, vấn đề ở đây là nhóm PROSPECTOR có thể dựa trên dữ liệu có giá trị về các mỏ tài nguyên đã biết và thông tin thống kê tin cậy. Các xác suất của từng sự kiện cũng được định nghĩa. Nhóm PROSPECTOR cũng có thể giả định sự độc lập có điều kiện của các chứng cứ, một ràng buộc phải thoả mãn để áp dụng được tiếp cận *Bayes*.

Tuy nhiên, trong nhiều lĩnh vực có thể áp dụng hệ chuyên gia, thông tin thống kê đáng tin cậy không có sẵn hoặc ta không thể giả định sự độc lập có điều kiện giữa các chứng cứ. Kết quả là nhiều nhà nghiên cứu thấy rằng phương pháp *Bayes* không thích hợp. Ví dụ, *Shortlife* và *Buchanan* không thể dùng tiếp cận xác suất cổ điển trong hệ thống MYCIN vì lĩnh vực y tế thường không cung cấp đủ dữ liệu. Việc không thoả các điều kiện này thúc đẩy sự phát triển của lý thuyết hệ số chắc chắn.

Dù cách tiếp cận hệ số chắc chắn thiếu nền tảng toán học như của lý thuyết xác suất, nó có vẻ tốt hơn suy diễn *Bayes* chủ quan trong những lĩnh vực như chẩn đoán, đặc biệt trong y tế. Trong các hệ chẩn đoán như MYCIN, các luật và hệ số chắc chắc hình thành do tri thức của chuyên gia và phán đoán chủ quan của họ. Hệ số chắc chắn được sử dụng trong những trường hợp không biết xác suất hoặc quá khó hoặc quá đắt để tính được xác suất. Cơ chế suy diễn chứng cứ có thể quản lý được chứng cứ được thu thập ngày càng nhiều lên, sự kết hợp và tách rời của các giả thuyết, cũng như các chứng cứ với mức độ tin cậy khác nhau. Bên cạnh đó, tiếp cận hệ số chắc chắn cung cấp sự giải thích tốt hơn về các tình huống điều khiển thông qua một hệ chuyên gia dựa trên luật.

Tiếp cận *Bayes* và hệ số chắc chắn là khác nhau, nhưng chúng cùng có một vấn đề chung: tìm được một chuyên gia có thể định lượng được thông tin cá nhân, chủ quan và có tính định lượng. Con người dễ mắc

sai lầm và do đó việc chọn một kỹ thuật quản lý sự không chắc chắn phụ thuộc lớn vào chuyên gia hiện có của lĩnh vực.

Phương pháp *Bayes* thích hợp nhất nếu dữ liệu thống kê đáng tin cậy tồn tại, kỹ sư tri thức có thể nắm bắt được và chuyên gia sẵn sàng cho những tình huống phân tích-quyết định nghiêm trọng. Nếu thiếu một trong những điều kiện trên, tiếp cận *Bayes* có thể hơi sai lệch để cho ra những kết qua có ý nghĩa. Một điểm nữa đáng chú ý là việc lan truyền độ tin cậy *Bayes* có độ phức tạp lũy thừa và do đó không thực tế đối với các cơ sở tri thức lớn.

Kỹ thuật hệ số chắc chắn, dù thiếu nền tảng hình thức nhưng vẫn đưa ra tiếp cận đơn giản để giải quyết các vấn đề không chắc chắn của các hệ chuyên gia và cho ra kết quả chấp nhận được trong nhiều ứng dụng.

VI. KẾT CHƯƠNG

Chương này giới thiệu hai phương pháp quản lý sự không chắc chắn trong các hệ thống tri thức. Một số điểm quan trọng của chương bao gồm:

- Sự không chắc chắn là sự thiếu vắng các chứng cứ chính xác cho phép ta đạt được một kết luận hoàn hảo đáng tin cậy. Nguồn gốc chính của tri thức không chắc chắn là do sự ngu dốt, nhập nhằng về ngôn ngữ biểu diễn.
- Lý thuyết xác suất cung cấp một cách tiếp cận chính xác và đúng đắn về mặt toán học để quản lý sự không chắc chắn. Các biểu thức xác suất bao gồm xác xuất tiền nghiệm và xác xuất có điều kiện trên các mệnh đề đơn giản và phức tạp.
- Phân phối xác suất kết hợp xác định xác suất của mỗi phép gán hoàn chỉnh các giá trị vào các biến ngẫu nhiên. Bảng này thường quá lớn để xây dựng và sử dụng ở dạng rõ ràng. Khi đã có bảng phân phối kết hợp đầy đủ, bảng này có thể được sử dụng để trả lời các câu hỏi bằng cách đơn giản cộng dồn các mục cho các sự kiện nguyên tố ứng với các mệnh đề câu hỏi.

- Luật Bayes cho phép tính các xác suất chưa biết từ các xác suất có điều kiện đã biết. Để có thể sử dụng luật Bayes, điều kiện độc lập có điều kiện giữa các chứng cứ phải được thoả mãn. Chúng ta cũng cần có những dữ liệu thống kê đáng tin cậy và định nghĩa các xác suất tiên nghiệm cho mỗi giả thiết.
- Hệ số chắc chắn là một phương án thay thế phổ biến của suy diễn Bayes. Các nguyên tắc cơ bản của lý thuyết này được giới thiệu trong MYCIN, một hệ chuyên gia chẩn đoán y khoa.
- Các hệ số chắc chắn được sử dụng nếu xác suất là không biết hoặc không thể dễ dàng thu được. Lý thuyết chắc chắn có thể quản lý các chứng cứ được thu thập dần dần, sự kết hợp và tách rời của các giả thiết cũng như các chứng cứ với mức độ tin cậy khác nhau.
- Cả suy diễn Bayes và lý thuyết chắc chắn đều có một vấn đề chung: tìm được một chuyên gia có khả năng lượng hóa các thông tin chủ quan và định tính.

VII.THUẬT NGỮ TIẾNG ANH

Sau đây là những thuật ngữ tiếng Anh liên quan đến những khái niệm đã dùng ở chương này:

atomic event: sự kiện nguyên tố

Bayesian reasoning: suy diễn Bayes

Bayes's rule: luật Bayes

certainty factor: hệ số chắc chắn

conditional independence: độc lập có điều kiện

conditional probability: xác suất có điều kiện

degree of belief: mức độ tin cậy

fuzzy logic: logic mờ

marginalization: biên hoá

measure of belief: độ đo tin cậy

measure of disbelief: độ đo không tin cậy

probability: xác suất

prior probability: xác suất tiền nghiệm

posterior probability: xác suất hậu nghiệm

probability distribution: phân phối xác suất

random variable: biến ngẫu nhiên

summing out: tính tổng ngoài

uncertainty: sự không chắc chắn

unconditional probability: xác suất không điều kiện

Phần 3
HỌC MÁY

Chương 8
GIỚI THIỆU HỌC MÁY

I. HỌC MÁY LÀ GÌ?

I.1 Các động lực của học máy

Một trong những yếu tố làm cho con người khác với các loài động vật khác đó chính là khả năng học. Tầm quan trọng của học không có gì phải bàn cãi, đây chính là nhân tố tạo thành các hành vi thông minh của con người. Trong lĩnh vực tìm kiếm các thuật toán có được trí thông minh như con người, một trong những vấn đề quan trọng là dạy cho máy biết cách học. Mặc dù đây là một vấn đề khó khăn, nhưng những thành công cũng như tiến bộ gần đây trong thuật toán và lý thuyết học máy đã cho thấy khả năng xây dựng các chương trình có khả năng học trên nhiều lĩnh vực trong thực tế. Ý tưởng đằng sau việc học đó chính là các giác quan không nên chỉ dùng để hành động, mà còn giúp nâng cao khả năng phản xạ trong tương lai. Việc học xảy ra thông qua quá trình tương tác với thế giới và quá trình ra quyết định.

Chương này giới thiệu về học máy, cụ thể đi vào một số phương pháp học cơ bản.

I.2 Các loại học

Các nhà nghiên cứu học máy đưa ra rất nhiều cách học khác nhau. Việc thiết kế thành phần học ảnh hưởng bởi 3 vấn đề chính sau:

- Những thành phần nào cần được học.
- Phản hồi nào được cung cấp để học những thành phần này.

- Những thành phần này được thể hiện như thế nào?

Ta sẽ lần lượt phân tích từng vấn đề một. Đầu tiên, các thành phần của hệ thống gồm:

1. Ánh xạ trực tiếp từ điều kiện của trạng thái hiện tại sang hành động.
2. Phương tiện để suy dẫn những tính chất liên quan của thế giới từ chuỗi quan sát.
3. Thông tin về cách thế giới phát triển và về kết quả của những hành động mà hệ thống có thể thực hiện.
4. Thông tin hữu ích về độ tốt của trạng thái.
5. Thông tin về giá trị của hành động cho biết mức độ phù hợp của hành động.
6. Đích mô tả nhóm các trạng thái mà hệ thống đạt được kết quả lớn nhất.

Mỗi thành phần có thể được học với phản hồi phù hợp. Xét ví dụ về việc huấn luyện hệ thống trở thành tài xế taxi. Mỗi lần người hướng dẫn kêu: "Thắng!" thì hệ thống có thể được phản xạ có điều kiện biết khi nào cần thắng (thành phần 1). Bằng cách xem nhiều hình ảnh có xe buýt, hệ thống có thể học để nhận biết chúng (2). Bằng cách thử hành động và quan sát kết quả chẳng hạn như thắng gấp trên đường ướt, thì hệ thống có thể học được hệ quả của hành động này (3). Sau đó, khi nó không nhận được tiền thưởng từ hành khách ngồi trên xe sau chuyến đi bị thắng gấp thường xuyên, hệ thống có thể học được thành phần tính toán hiệu quả tổng thể (4).

Loại phản hồi được cung cấp cho việc học thường là nhân tố quan trọng giúp xác định bản chất của thuật toán học. Trong lĩnh vực học máy thường phân biệt ba loại học: giám sát, không giám sát và tăng cường.

Học có giám sát

Học có giám sát học ánh xạ từ các mẫu đầu vào và đầu ra. Trường hợp (1), (2) và (3) đều là ví dụ của học có giám sát.

Học không giám sát

Học không giám sát liên quan tới việc học từ đầu vào trong khi không có giá trị đầu ra cụ thể nào được cung cấp. Ví dụ, hệ thống taxi hoàn toàn có thể hình thành khái niệm về "ngày giao thông tốt" và "ngày giao thông xấu", mà không hề cần đến mẫu đã gán nhãn cho những khái niệm này. Hệ thống học không giám sát thuần túy không thể học cách làm, vì nó không có thông tin cũng như nhận biết được một hành động đúng hoặc trạng thái phù hợp.

Học tăng cường

Học tăng cường tổng quát nhất trong cả ba loại học. Thay vì nghe theo những hướng dẫn từ trước, hệ thống học tăng cường sẽ tự học từ sự tăng cường. Học tăng cường thường chứa bài toán con, học cách hoạt động của môi trường.

II. NAÏVE *BAYES* – MỘT MÔ HÌNH HỌC ĐƠN GIẢN

II.1 Ví dụ học máy

Xét một ví dụ đơn giản sau đây để minh họa cho mô hình học đơn giản *Bayes*. Ví dụ được trình bày trong Bảng 8-1:

Bảng 8-1. *Một ví dụ học máy đơn giản*

f_1	f_2	f_3	f_4	y
0	1	1	0	1
0	0	1	1	1
1	0	1	0	1
0	0	1	1	1
0	0	0	0	1
1	0	0	1	0
1	1	0	1	0
1	0	0	0	0
1	1	0	1	0
1	0	1	1	0

Trong ví dụ này, các giá trị f_i là các giá trị đầu vào và ta cần dự đoán giá trị kết xuất y theo các giá trị này. Mô hình *Naïve Bayes* thực hiện

huấn luyện bằng cách thống kê các điểm dữ liệu. Ta thống kê các giá trị:

$R_i(f_i=j, y=k)$: tỷ lệ các mẫu lớp y=k có đặc trưng $f_i=j$.

Ví dụ:

- $R_1(1,1) = 1/5$: tỷ lệ tất cả các mẫu **dương(y=1)** có đặc trưng 1 = **1.**
- $R_1(0,1) = 4/5$: tỷ lệ tất cả các mẫu **dương** có đặc trưng 1 = 0.

Với các thống kê như vậy, ta tính được toàn bộ các con số R cho tập dữ liệu:

$R_1(1,1) = 1/5$ $R_1(0,1) = 4/5$
$R_1(1,0) = 5/5$ $R_1(0,0) = 0/5$
$R_2(1,1) = 1/5$ $R_2(0,1) = 4/5$
$R_2(1,0) = 2/5$ $R_2(0,0) = 3/5$
$R_3(1,1) = 4/5$ $R_3(0,1) = 1/5$
$R_3(1,0) = 1/5$ $R_3(0,0) = 4/5$
$R_4(1,1) = 2/5$ $R_4(0,1) = 3/5$
$R_4(1,0) = 4/5$ $R_4(0,0) = 1/5$

Với dữ liệu mới đưa vào, ví dụ x=<0, 0, 1, 1>, lớp của mẫu dữ liệu được quyết định bằng việc so sánh hai số:

$S(1) = R_1(0,1) \times R_2(0,1) \times R_3(1,1) \times R_4(1,1) = 0.205.$

$S(0) = R_1(0,0) \times R_2(0,0) \times R_3(1,0) \times R_4(1,0) = 0.$

Do: S(1) > S(0), mẫu mới được dự đoán rơi vào lớp 1. Ta có thể thấy phương pháp học *Bayes* khá đơn giản, chỉ bằng các thống kê tỷ lệ xuất hiện của các thuộc tính trên các lớp và sau đó dự đoán bằng cách so sánh tích những tỷ lệ này. Tuy nhiên, đằng sau những bước đơn giản này là một nền tảng vững chắc vốn được thiết lập từ hàng thế kỷ trước: *lý thuyết xác suất*. Những phần tiếp theo sẽ trình bày chi tiết về việc áp dụng mô hình xác suất vào bài toán học máy.

II.2 Mô hình xác suất

Ta xem xét mối quan hệ giữa việc học có giám sát và suy diễn theo *Bayes*. Xét ví dụ về cách thiết kế thuật toán học dựa trên luật *Bayes*.

Ở ví dụ học có giám sát, ta muốn xấp xỉ hàm mục tiêu chưa biết f: $X \rightarrow Y$ hoặc tương đương với $P(Y \mid X)$. Giả sử Y là biến giá trị nhị phân ngẫu nhiên và X là một vector chứa n thuộc tính nhị phân. Nói cách khác, $X = \langle X_1, X_2, ..., X_n \rangle$ với X_i là biến nhị phân ngẫu nhiên đại diện cho thuộc tính thứ i của X.

Áp dụng luật *Bayes*, ta thấy rằng $P(Y = y_i | X)$ có thể thể hiện như sau:

$$P(Y = y_i | X = x_k) = \frac{P(X = x_k | Y = y_i) P(Y = y_i)}{\sum_j P(X = x_k | Y = y_i) P(Y = y_i)}$$

Với y_m thể hiện giá trị thứ m có thể của Y và tổng trong mẫu số là trên toàn bộ giá trị hợp lệ của biến ngẫu nhiên Y.

Một cách để học $P(Y|X)$ là dùng dữ liệu huấn luyện để ước lượng $P(X|Y)$ và $P(Y)$. Ta có thể dùng ước lượng trên cùng với luật *Bayes* để xác định $P(Y \mid X = x_k)$ cho bất kì thể hiện x_k mới nào.

Nếu muốn huấn luyện bộ phân lớp *Bayes* bằng cách ước lượng $P(X|Y)$ và $P(Y)$ thì câu hỏi đặt ra là cần bao nhiêu dữ liệu huấn luyện để có được ước lượng tin cậy cho phân bố đó, giả sử mẫu huấn luyện được phát sinh ngẫu nhiên từ phân bố chưa biết P(X), sau đó gán nhãn mẫu này với giá trị Y của nó.

Hàng trăm mẫu huấn luyện phát sinh độc lập đủ để có được **ước lượng khả suất tối đại** cho $P(Y)$ trong vòng vài phần trăm sai số với Y là một biến nhị phân. Tuy nhiên, ước lượng một cách chính xác $P(X|Y)$ thường cần nhiều mẫu hơn. Để hiểu tại sao như vậy ta xem xét số lượng tham số mà ta phải ước lượng khi Y là giá trị nhị phân và X là một vector gồm n thuộc tính nhị phân. Trong trường hợp này, ta cần ước lượng tập tham số:

$$\theta_{ij} \equiv P(X = x_i \mid Y = y_j)$$

Với chỉ số i nhận 2^n giá trị có thể (cho mỗi giá trị vector có thể của X) và j nhận 2 giá trị có thể. Do đó ta cần ước lượng xấp xỉ 2^{n+1} tham số. Để tính toán chính xác số lượng tham số cần tính thì tổng theo i của θ_{ij} phải bằng một (với bất kì j cố định nào). Do đó, với bất kì giá trị y_j nào và 2^n giá trị x_i có thể, ta chỉ cần tính $2^n - 1$ tham số độc lập. Với hai giá trị có thể của Y, ta phải ước lượng tổng cộng $2(2^n - 1)$ tham số θ_{ij}

như vậy. Hơn nữa, để có được ước lượng tin cậy cho mỗi tham số này, ta sẽ cần quan sát từng thể hiện khác nhau này nhiều lần. Điều này rõ ràng không khả thi trong hầu hết ứng dụng học trong thực tế.

II.3 Luật *Bayes*

Yêu cầu ở đây là tìm cách giảm độ phức của thuật toán học phân lớp *Bayes*. Bộ phân lớp *Naïve Bayes* thực hiện điều này bằng cách đưa ra giả thuyết độc lập điều kiện làm giảm số lượng tham số sẽ ước lượng từ $2(2^n - 1)$ xuống còn $2n$ khi mô hình hóa $P(X|Y)$.

Độc lập có điều kiện

Định nghĩa: Cho các biến ngẫu nhiên X, Y và Z, ta nói X là độc lập có điều kiện của Y cho trước Z nếu và chỉ nếu phân bố xác suất trên X độc lập so với giá trị của Y cho trước Z, đó là:

$$(\forall i, j, k)\, P\left(X = x_i | Y = y_j, Z = z_k\right) = P(X = x_i \mid Z = z_k)$$

Ví dụ, xét 3 biến nhị phân ngẫu nhiên mô tả thời tiết hiện tại: Mưa, Sấm và Chớp. Ta có thể suy diễn rằng Sấm độc lập với Mưa trong điều kiện có Chớp. Bởi vì ta biết rằng Chớp tạo ra Sấm, khi ta biết có Chớp thì không có thông tin gì thêm về Sấm được cung cấp bởi Mưa. Đương nhiên là Sấm có sự phụ thuộc vào giá trị của Mưa, nhưng sẽ không có sự phụ thuộc nào khi ta biết giá trị của Chớp

Thuật toán *Naïve Bayes* là một thuật toán phân lớp dựa trên luật *Bayes*, trong đó giả sử rằng thuộc tính $X_1, \ldots, X_n$ đều độc lập có điều kiện lẫn nhau đối với Y. Giá trị của giả sử này nằm ở chỗ nó đơn giản hóa thể hiện của $P(X|Y)$ và vấn đề ước lượng nó từ dữ liệu huấn luyện. Xem xét trường hợp mà: $X = \langle X_1, X_2 \rangle$.

$$P(X|Y) = P(X_1, X_2|Y) = P(X_1|X_2, Y)\, P(X_2|Y) = P(X_1|Y)\, P(X_2|Y)$$

Trong đó, biểu thức thứ ba suy từ tính chất tổng quát của xác suất và biểu thức thứ tư suy trực tiếp từ định nghĩa độc lập có điều kiện ở trên. Tổng quát hơn, khi X chứa n thuộc tính đôi một độc lập có điều kiện cho trước Y thì ta có :

$$P(X_1 \dots X_2|Y) = \prod_{i=1}^{n} P(X_i|Y) \quad (1)$$

Lưu ý rằng khi Y và X_i là biến nhị phân, ta chỉ cần $2n$ tham số để định nghĩa $P(X_i = x_{ik}|Y=y_j)$ cho các giá trị i, j, k. Đây là một sự giảm thiểu đáng kể so với $2(2^n - 1)$ tham số cần để đặc trưng hóa $P(X|Y)$ nếu ta không giả sử độc lập có điều kiện xảy ra.

Bây giờ ta mở rộng thuật toán Naïve *Bayes*, giả sử rằng Y là biến giá trị rời rạc bất kì và các thuộc tính $X_1,\dots, X_n$ là các thuộc tính giá trị rời rạc hoặc giá trị thực bất kì. Mục đích là huấn luyện bộ phân lớp sao cho nó cho ra phân bố xác suất của các giá trị của Y, với mỗi thể hiện X mới ta yêu cầu nó phân lớp. Biểu thức xác suất trong đó Y sẽ nhận giá trị có thể thứ k như sau (dựa trên luật *Bayes*):

$$P(Y = y_k|X_1 \dots X_n) = \frac{P(Y = y_k)P(X_1 \dots X_n|Y = y_k)}{\sum_j P(Y = y_j)P(X_1 \dots X_n|Y = y_j)}$$

Trong đó tổng trên tất cả giá trị y_j có thể của Y. Bây giờ, giả sử X_i độc lập có điều kiện với Y, ta có thể sử dụng công thức (1) viết lại như sau:

$$P(Y = y_k|X_1 \dots X_n) = \frac{P(Y = y_k)\prod_{i=1}^{n} P(X_i|Y_k)}{\sum_j P(Y = y_j)\prod_{i=1}^{n} P(X_i|Y_j)} \quad (2)$$

Công thức (2) là công thức cơ bản của bộ phân lớp *Naïve Bayes*. Với một thể hiện mới $X^{new} = (X_1, \dots, X_n)$ công thức này cho ta thấy cách tính xác suất mà Y nhận bất kì giá trị nào cho trước, với giá trị thuộc tính X^{new} và với phân bố $P(Y)$ và $P(X|Y)$ được ước lượng từ dữ liệu huấn luyện. Nếu ta chỉ quan tâm đến giá trị nhiều khả năng nhất của Y thì ta có luật phân lớp *Naïve Bayes* sau:

$$Y \leftarrow \text{argmax}_{y_k} \frac{P(Y = y_k)\prod_{i=1}^{n} P(X_i|Y_k)}{\sum_j P(Y = y_j)\prod_{i=1}^{n} P(X_i|Y_j)}$$

Biểu thức đơn giản (vì mẫu số không phụ thuộc vào y_k):

$$Y \leftarrow \text{argmax}_{y_k} P(Y = y_k)\prod_{i=1}^{n} P(X_i|Y_k)$$

II.4 Phép sửa Laplace

Khi n thuộc tính đầu vào X_i, mỗi thuộc tính nhận J giá trị rời rạc có thể, và Y là biến rời rạc nhận K giá trị có thể thì công việc học sẽ là ước lượng hai bộ tham số này. Bộ đầu tiên là:

$$\theta_{ijk} \equiv P(X = x_{ij} \mid Y = y_k)$$

Ứng với mỗi thuộc tính đầu vào X_i, mỗi giá trị x_{ij} có thể và mỗi giá trị y_k có thể của Y. Lưu ý ta có nJK tham số như vậy và cũng lưu ý rằng chỉ có $n(J-1)K$ trong số đó là độc lập, với điều kiện là $1 = \Sigma_j \theta_{ijk}$ ứng với mỗi cặp giá trị i, k.

Hơn nữa, ta phải ước lượng tham số định nghĩa xác suất tiền điều kiện trên Y:

$$\pi_k \equiv P(Y = y_k)$$

Lưu ý ta có K tham số như vậy, trong đó K – 1 là độc lập.

Ta có thể ước lượng những tham số này sử dụng ước lượng khả suất tối đại (dựa trên việc tính toán tần suất của những sự kiện khác nhau trong dữ liệu).

Ước lượng khả suất tối đại cho θ_{ijk} với tập mẫu huấn luyện D cho bởi:

$$\hat{\theta}_{ijk} = \hat{P}\left(X = x_{ij}|Y = y_k\right) = \frac{\#D\{X_i = x_{ij} \wedge Y = y_k\}}{\#D\{Y = y_k\}}$$

Với toán tử #D{x} trả về số lượng phần tử trong tập D thỏa tính chất x.

Một điều nguy hiểm với ước lượng khả suất tối đại là nó có thể trả về kết quả ước lượng θ bằng 0, nếu dữ liệu không chứa bất kì mẫu huấn luyện nào thỏa mãn điều kiện. Để tránh điều này, ta thường dùng một ước lượng "trơn" trong đó thêm vào một số mẫu "ảo" bổ sung và với giả sử rằng những mẫu này được phân phối đồng đều trên các giá trị của X_i. Ước lượng trơn này được cho bởi công thức:

$$\hat{\theta}_{ijk} = \hat{P}\left(X = x_{ij} | Y = y_k\right) = \frac{\#D\{X_i = x_{ij} \wedge Y = y_k\} + l}{\#D\{Y = y_k\} + lJ}$$

Với J là số lượng giá trị khác nhau mà X_i có thể nhận và l xác định trọng số của việc làm trơn (số lượng mẫu ảo là lJ). Biểu thức này tương ứng với ước lượng MAP cho θ_{ijk} nếu ta giả sử tồn tại phân bố Dirichlet trên các tham số θ_{ijk} với những tham số có giá trị bằng nhau. Nếu l gán bằng 1 thì phương pháp này được gọi là phép sửa Laplace.

Ước lượng khả suất tối đại cho π là:

$$\hat{\pi}_k = \hat{P}(Y = y_k) = \frac{\#D\{Y = y_k\}}{|D|}$$

Với |D| là số lượng phần tử trong tập huấn luyện D.

$$\hat{\pi}_k = \hat{P}(Y = y_k) = \frac{\#D\{Y = y_k\} + l}{|D| + lK}$$

Với K là số lượng giá trị khác nhau mà Y có thể nhận và l xác định trọng số của việc làm trơn đối với các mẫu ảo có liên quan trên dữ liệu quan sát D.

III. HỌC CÂY QUYẾT ĐỊNH

Học cây quyết định là một trong những thuật toán học đơn giản nhất và cũng là dạng thành công nhất. Nó đóng vai trò khá tốt trong việc giới thiệu về thuật toán học và nó cũng dễ cài đặt. Đầu tiên ta tìm hiểu thành phần thực thi và sau đó xem cách nó học.

III.1 Cây quyết định

Cây quyết định nhận đầu vào là một đối tượng hoặc tình huống mô tả bởi tập các thuộc tính và trả về "quyết định" - dự đoán về giá trị đầu ra ứng với đầu vào. Các thuộc tính đầu vào có thể rời rạc hoặc liên tục. Từ đây ta giả sử đầu vào là rời rạc. Giá trị đầu ra cũng có thể là rời rạc hoặc liên tục, việc học một hàm giá trị rời rạc gọi là học để **phân lớp**, còn học một hàm có giá trị liên tục gọi là học **hồi qui**. Ta sẽ tập trung về bài toán phân lớp nhị phân, trong đó mỗi mẫu được phân loại là đúng hoặc sai.

Cây quyết định cho ra quyết định qua một loạt các bước kiểm tra. Mỗi nút trong cây tương ứng với một bước kiểm tra giá trị của một trong các thuộc tính và các nhánh đi ra từ nút được gán nhãn bằng giá trị có thể của thuộc tính được kiểm tra. Mỗi nút lá của cây xác định giá trị được trả về khi duyệt tới nút lá. Một ví dụ minh họa đơn giản, xét vấn đề có nên đợi để có bàn ở một nhà hàng không. Mục đích ở đây là học định nghĩa cho hàm mục tiêu *Sẽ Đợi*. Để trở thành một bài toán học, ta phải xác định có những thuộc tính nào mô tả mẫu trong miền. Giả sử ta sẽ quyết định dựa trên những thuộc tính sau:

1. *Lựa chọn*: Có nhà hàng nào khác gần đó phù hợp không?
2. *Quầy Bar*: Nhà hàng có một quầy bar thoải mái để ngồi đợi không?
3. *Cuối tuần*: Đúng (true) nếu vào thứ bảy và chủ nhật.
4. *Đói*: Ta có đói bụng không?
5. *Khách*: Có bao nhiêu người trong nhà hàng (với giá trị là Không, Ít và Đầy).
6. *Giá*: Giá cả ở nhà hàng.
7. *Trời mưa*: Trời có đang mưa không?
8. *Đặt chỗ*: Có đặt chỗ trước không?
9. *Loại*: Loại nhà hàng (Pháp, Ý, Thái hoặc đồ ăn nhanh).
10. *Đợi bao lâu*: Khoảng thời gian chờ ước lượng bởi nhà hàng (0 -10 phút, 10-30, 30-60, > 60).

Cây quyết định thường được dùng được thể hiện trong Hình 8.1.

Lưu ý rằng cây này không dùng tới thuộc tính *Giá* và *Loại*, thực ra là xem như chúng không có gì liên quan. Các mẫu được xử lý bởi cây bắt đầu từ nút gốc và đi theo nhánh phù hợp cho tới khi đạt tới nút lá.

Ví dụ, một mẫu với *Khách = Đầy* và *Đợi bao lâu = 0 -10* sẽ được phân vào lớp dương (= ta sẽ đợi).

III.2 Tính biểu diễn của cây quyết định

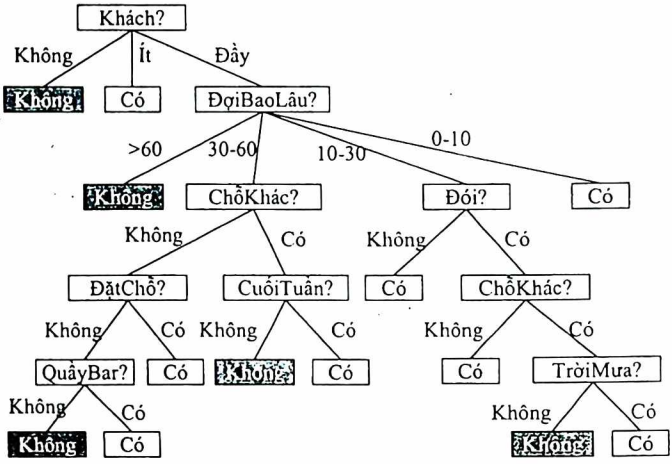
Bất kì cây quyết định giả thuyết cho việc dự đoán *Sẽ Đợi* hay không có thể được xem như một xác nhận với dạng:

$$\forall s \; \text{SẽĐợi}(s) \Leftrightarrow (P_1(s) \vee P_2(s) \vee ... \vee P_n(s)).$$

Với mỗi điều kiện $P_i(s)$ là liên kết của phép kiểm tra tương ứng với đường đi từ nút gốc của cây tới nút lá có kết quả đúng. Mặc dù, biểu thức này trông giống một câu bậc nhất, theo nghĩa nào đó thì nó là mệnh đề vì nó chứa một biến và tất cả vị từ đều đơn. Cây quyết định thực sự mô tả mối quan hệ giữa *Sẽ Đợi* và tổ hợp logic của các giá trị của thuộc tính. Ta không thể nào sử dụng cây quyết định để thể hiện các kiểm tra liên quan tới hai hay nhiều đối tượng, ví dụ:

$$\exists r_2 \; \text{GầnĐây}(r_2, r) \wedge \text{Giá}(r, p) \wedge \text{Giá}(r_2, p_2) \wedge \text{RẻHơn}(p_2, p)$$

(Có nhà hàng nào gần đây rẻ hơn không?). Rõ ràng, ta có thể thêm thuộc tính nhị phân khác với tên là *NhàHàngRẻHơnỞGầnĐây*, nhưng khó mà cộng tất cả thuộc tính như vậy được.



Hình 8.1 - *Cây quyết định cho việc quyết định có đợi để có bàn không.*

Cây quyết định được thể hiện đầy đủ bởi ngôn ngữ mệnh đề, tức là bất kì hàm Boolean nào đều có thể được viết như một cây quyết định. Điều này có thể được thực hiện đơn giản bằng cách tạo một dòng trong bảng chân trị của hàm cho mỗi đường đi trong cây. Nhưng nó cũng sẽ tạo ra thể hiện rất lớn vì bảng chân trị sẽ có rất nhiều dòng. Điều ta mong muốn là xây dựng cây quyết định có thể đại diện cho nhiều hàm bằng cấu trúc cây nhỏ hơn.

III.3 Rút cây quyết định từ ví dụ

Một ví dụ về cây quyết định nhị phân gồm vector thuộc tính đầu vào X và một giá trị đầu ra nhị phân y. Tập các mẫu (X_1, y_1) ... (X_{12}, y_{12}) thể hiện trong Bảng 8-2.

Mẫu đúng là mẫu mà đích *Sẽ Đợi* ra *Có*$(X_1, X_3, ...)$; mẫu sai là mẫu mà nó ra *Không*$(X_2, X_5, ...)$. Tập mẫu đầy đủ được gọi tập huấn luyện.

Bài toán tìm kiếm một cây quyết định mà đúng trên tập huấn luyện có vẻ khó, nhưng trên thực tế có một lời giải đơn giản. Ta có thể đơn

giản xây dựng một cây quyết định mà nó có một đường đi đến nút lá cho mỗi mẫu, trong đó mỗi thuộc tính lần lượt được kiểm tra và dựa trên giá trị của mẫu và nút lá có mẫu phân lớp. Khi đưa vào lại mẫu đó, cây quyết định sẽ cho ra phân lớp đúng. Nhưng cây quyết định này sẽ không cho biết gì nhiều về những trường hợp khác chưa biết đến.

Bảng 8-2. *Các mẫu cho ví dụ nhà hàng. Các thuộc tính được ký hiệu theo thứ tự liệt kê trong phần III.1*

Mẫu	Các thuộc tính										Mục tiêu
	A1	*A2*	*A3*	*A4*	*A5*	*A6*	*A7*	*A8*	*A9*	*A10*	*Sẽ Đợi*
X_1	*Có*	*K*	*K*	*Có*	*It*	$	*K*	*Có*	*Pháp*	0-10	*Có*
X_2	*Có*	*K*	*K*	*Có*	*Đầy*	$	*K*	*K*	*Thái*	30-40	*K*
X_3	*K*	*Có*	*K*	*K*	*Ít*	$	*K*	*K*	*Nhanh*	0-10	*Có*
X_4	*Có*	*K*	*Có*	*Có*	*Đầy*	$	*Có*	*K*	*Thái*	10-30	*Có*
X_5	*Có*	*K*	*Có*	*K*	*Đầy*	$$$	*K*	*Có*	*Pháp*	>60	*K*
X_6	*K*	*Có*	*K*	*Có*	*Ít*	$$	*Có*	*Có*	*Ý*	0-10	*Có*
X_7	*K*	*Có*	*K*	*K*	*K*	$	*Có*	*K*	*Nhanh*	0-10	*K*
X_8	*K*	*K*	*K*	*Có*	*Ít*	$$	*Có*	*Có*	*Thái*	0-10	*Có*
X_9	*K*	*Có*	*Có*	*K*	*Đầy*	$	*Có*	*K*	*Nhanh*	>60	*K*
X_{10}	*Có*	*Có*	*Có*	*Có*	*Đầy*	$$$	*K*	*Có*	*Ý*	10-30	*K*
X_{11}	*K*	*K*	*K*	*K*	*K*	$	*K*	*K*	*Thái*	0-10	*K*
X_{12}	*Có*	*Có*	*Có*	*Có*	*Đầy*	$	*K*	*K*	*Nhanh*	30-60	*Có*

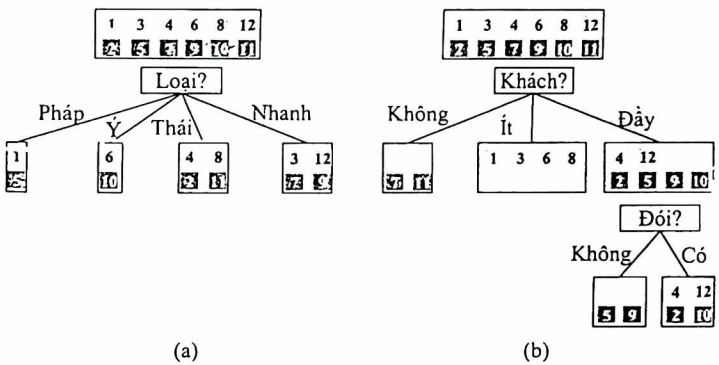
Vấn đề với cây này là nó chỉ ghi nhớ những gì quan sát được. Nó không rút trích bất kỳ dạng nào từ mẫu, do đó ta không thể mong nó có thể ngoại suy cho những mẫu mà nó chưa hề thấy qua. Áp dụng **phương pháp cắt tỉa của Ockham**, ta sẽ tìm ra cây quyết định nhỏ

nhất mà nó nhất quán với các mẫu. Việc tìm cây nhỏ nhất là một bài toán khó. Tuy nhiên, ta có thể dùng một vài *heuristic* đơn giản để thực hiện công việc tìm ra một cái "đủ nhỏ". Ý tưởng chính của thuật toán là thực hiện kiểm tra thuộc tính quan trọng nhất trước. "Quan trọng nhất" có nghĩa là cái mà tạo nên sự khác biệt lớn nhất khi phân lớp mẫu. Bằng cách này, ta hi vọng sẽ đạt được sự phân lớp chính xác với một số lượng phép thử nhỏ, điều này đồng nghĩa với việc tất cả đường đi trên cây sẽ ngắn và như vậy nguyên cả cây sẽ nhỏ.

Hình 8.2 cho thấy cách thuật toán chạy. Cho trước 12 mẫu huấn luyện, được phân thành tập đúng và sai. Sau đó ta quyết định thuộc tính nào sẽ được dùng cho phép thử đầu tiên trên cây. Hình 8.2 (a) cho thấy *Loại* là một thuộc tính xấu vì nó cho ta 4 đầu ra có thể có, mỗi cái có cùng số lượng mẫu đúng và sai. Ngược lại, trong Hình 8.2 (b) ta thấy rằng *Khách* là một thuộc tính khá quan trọng bởi vì nếu giá trị của nó là *Không* hoặc *Ít* thì ta được các tập mẫu mà có thể trả lời chắc chắn (lần lượt là *Không* và *Có*). Nếu giá trị là *Đầy*, ta được tập mẫu lẫn lộn. Một cách tổng quát, sau khi thuộc tính thử đầu tiên tách các mẫu ra, mỗi đầu ra sẽ là một cây quyết định mới học với số mẫu và thuộc tính ít hơn. Có bốn trường hợp cần xem xét:

1. Nếu ta có một số mẫu đúng và một số mẫu sai, thì ta chọn thuộc tính phân tách chúng tốt nhất. Hình 8.2 (b) cho thấy *Đói* được chọn để tách các mẫu.
2. Nếu tất cả mẫu còn lại là đúng (hoặc sai) thì ta dừng ở đây. Ta có thể trả lời *Có* hoặc *Không*. Hình 8.2 (b) cho thấy một ví dụ trong trường hợp của *Không* và *Ít*.
3. Nếu không còn mẫu nào thì điều này có nghĩa không có mẫu nào như vậy được quan sát và ta trả về giá trị mặc định được tính từ phân lớp đa số ở nút cha.
4. Nếu không còn thuộc tính nào, nhưng vẫn còn cả mẫu đúng và sai thì có vấn đề ở đây. Nghĩa là xảy ra trường hợp các mẫu có cùng mô tả nhưng được phân lớp khác nhau. Điều này xảy ra khi dữ liệu nào đó bị sai, ta gọi đây là nhiễu trong dữ liệu. Nó cũng xảy ra khi các thuộc tính không cung cấp đủ thông tin hoặc không xác định. Cách đơn giản để giải quyết vấn đề là bỏ phiếu theo số đông.

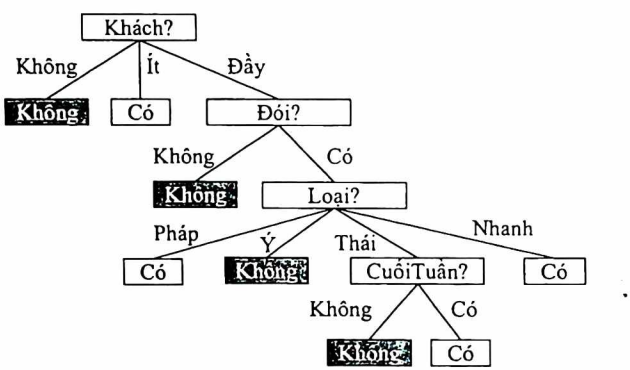
Cây được tạo bởi thuật toán áp dụng trên 12 mẫu dữ liệu được thể hiện trong Hình 8.4. Cây này hoàn toàn khác với cây ban đầu trong Bảng 8-2, dù dữ liệu thực sự được phát sinh bằng cây gốc này. Một nhận xét có thể rút ra là thuật toán học chưa làm tốt công việc học. Tuy nhiên, đây có thể là một kết luận sai. Thuật toán học chỉ xem xét đến các mẫu, (xem Hình 8.4) nó không những đúng với tất cả các mẫu mà còn đơn giản hơn cây gốc. Thuật toán học không kiểm tra thuộc tính *Mưa* và *Đặt chỗ* vì nó có thể phân loại tất cả mẫu mà không cần đến hai thuộc tính này.



Hình 8.2 - *Tách các mẫu bằng việc kiểm tra thuộc tính. (a) Tách trên Loại không cho ta sự phân biệt giữa mẫu đúng và mẫu sai. (b) Tách trên Khách lại làm tốt công việc phân biệt mẫu đúng và mẫu sai. Sau khi tách trên Khách, kiểm tra tiếp Đói sẽ cho kết quả khá tốt.*

```
function Học_Cây_Quyết_Định(tập_mẫu, thuộc_tính, mặc_định) return cây
quyết định t
    if tập_mẫu rỗng then return mặc_định
    else if tất cả tập_mẫu đều thuộc cùng một lớp then return lớp
    else if thuộc_tính rỗng then return Lớp_Đa_Số(tập_mẫu)
    else
      b ←Chọn_Thuộc_Tính(thuộc_tính, tập_mẫu)
      t ← cây quyết định mới với gốc là b
      m ←Giá_Trị_Đa_Số(tập_mẫu)
      for each vi trong b do
            tập_mẫu_i ← {phần tử trong tập_mẫu với b = vi}
            st ←Học_Cây_Quyết_Định (tập_mẫu_i, thuộc_tính_tốt_nhất,
    mặc_định)
            Thêm một nhánh vào t với nhãn v_i và cây con là st
      return t
```

Hình 8.3- *Thuật toán xây dựng cây định danh.*



Hình 8.4 - *Cây quyết định rút ra từ tập huấn luyện gồm 12 mẫu.*

III.4 Chọn thuộc tính kiểm tra

Phương pháp chọn thuộc tính sử dụng trong học cây quyết định được thiết kế để tối tiểu độ sâu của cây. Ý tưởng là chọn thuộc tính đạt được

sự phân lớp các mẫu càng chính xác càng tốt. Một thuộc tính hoàn hảo sẽ chia các mẫu thành các tập chỉ gồm mẫu đúng hoặc mẫu sai. Thuộc tính *Khách* không hoàn hảo nhưng lại khá tốt. Thuộc tính vô dụng để lại tập mẫu với tỉ lệ đúng và sai gần như ban đầu. Ta cần một cách tính hình thức cho khái niệm "khá tốt" và "vô dụng", có thể cài đặt hàm chọn Chọn_Thuộc_Tính. Công thức tính phải đạt cực đại khi thuộc tính hoàn hảo và cực tiểu khi thuộc tính không có ích gì cả. Một đại lượng tính phù hợp đó là lượng thông tin được cung cấp bởi thuộc tính, một khái niệm mang tính toán học định nghĩa bởi *Shannon* và *Weaver* năm 1949. Để hiểu khái niệm thông tin, xem nó như câu trả lời cho câu hỏi như khi nào đồng xu sẽ xuất hiện mặt ngửa. Lượng thông tin chứa trong câu trả lời phụ thuộc vào tri thức tiền định. Biết càng ít, thì thông tin được cung cấp sẽ càng nhiều. Lý thuyết thông tin đo lượng thông tin bằng số bit. Một bit thông tin đủ để trả lời câu hỏi đúng/sai về điều mà ta không biết trước chẳng hạn mặt của đồng xu được tung. Tổng quát, nếu những câu trả lời có thể v_i có xác suất $P(v_i)$ thì lượng thông tin I của câu trả lời thực sự được cho bởi:

$$I\left(P(v_1),\ldots,P(v_n)\right)=\sum_{i=1}-P(v_i)log_2P(v_i)$$

Để kiểm tra công thức này, với việc tung đồng xu, ta có:

$$I\left(\frac{1}{2},\frac{1}{2}\right)=-\frac{1}{2}log_2\frac{1}{2}-\frac{1}{2}log_2\frac{1}{2}=1bit$$

Nếu biết đồng xu có 99% xấp, ta có I(1/100, 99/100) = 0.08 bit và với xác suất xấp đạt 1, thông tin về câu trả lời thực sự sẽ là 0.

Với bài toán học cây quyết định, câu hỏi cần được trả lời là với một mẫu thì cái nào là phân lớp dung của nó? Một cây quyết định đúng sẽ trả lời được câu hỏi này. Một ước lượng xác suất của câu trả lời trước khi bất kì thuộc tính nào được kiểm tra sẽ là tỉ lệ giữa mẫu đúng và sai trong tập huấn luyện. Giả sử tập huấn luyện chứa *p* mẫu đúng và *n* mẫu sai. Khi đó ước lượng thông tin chứa trong câu trả lời đúng là:

$$I\left(\frac{p}{p+n},\frac{n}{p+n}\right)=-\frac{p}{p+n}log_2\frac{p}{p+n}-\frac{n}{p+n}log_2\frac{n}{p+n}$$

Tập huấn luyện trong Bảng 8-2 có $p = n = 6$ nên ta cần 1 bit thông tin.

Phép thử trên một thuộc A sẽ không cho ta biết nhiều thông tin, nhưng nó sẽ cho ta một con số. Ta có thể đo chính xác bằng cách: xem ta còn cần bao nhiêu thông tin khi kiểm tra thuộc tính. Thuộc tính A bất kì chia tập huấn luyện E thành các tập con $E_1,\ldots, E_v$ tương ứng với các giá trị của A, A có v giá trị khác nhau. Mỗi tập con E_i có p_i mẫu đúng và n_i mẫu sai, nên nếu ta đi theo nhánh này thì ta sẽ cần thêm $I(p_i/(p_i + n_i), n_i/(p_i + n_i))$ bit thông tin để trả lời câu hỏi. Một mẫu được chọn ngẫu nhiên từ tập huấn luyện với thuộc tính đó có giá trị thứ I với xác suất chọn $(p_i + n_i)/(p + n)$, sau khi kiểm tra thuộc tính A, trung bình ta cần:

$$R(A)=\sum_{i=1}^{v}\frac{p_i+n_i}{p+n}I\left(\frac{p_i}{p+n},\frac{n_i}{p+n}\right)$$

Bit thông tin để phân lớp mẫu này. **Độ lợi thông tin** từ việc kiểm tra thuộc tính là sự khác biệt giữa yêu cầu thông tin ban đầu và yêu cầu mới:

$$Gain(A)=I\left(\frac{p}{p+n},\frac{n}{p+n}\right)-R(A)$$

Heuristic được sử dụng trong hàm Chọn_Thuộc_Tính chỉ để chọn thuộc có độ lợi thông tin lớn nhất. Quay lại thuộc tính ta xem xét trong Hình 8.2, ta có:

$$Gain(Khách)=1-\left[\frac{2}{12}I(0,1)+\frac{4}{12}I(1,0)+\frac{6}{12}I\left(\frac{2}{6},\frac{4}{6}\right)\right] \approx 0.541 bit$$

$$Gain(Loai)=1-\left[\frac{2}{12}I\left(\frac{1}{2},\frac{1}{2}\right)+\frac{2}{12}I\left(\frac{1}{2},\frac{1}{2}\right)+\frac{4}{12}I\left(\frac{2}{4},\frac{2}{4}\right)+\frac{4}{12}I\left(\frac{2}{4},\frac{2}{4}\right)\right]=0$$

Khẳng định nhận xét rằng *Khách* là thuộc tính phân tách tốt hơn. Trên thực tế, *Khách* có độ lợi thông tin cao nhất trong số các thuộc tính và sẽ được chọn làm nút gốc bởi thuật toán học cây quyết định.

III.5 Đánh giá hiệu quả của thuật toán học

Một thuật toán học tốt nếu nó tạo ra các giả thuyết làm tốt công việc dự đoán lớp của mẫu chưa được học. Ta sẽ xem xét phương pháp đánh giá chất lượng của việc dự đoán.

Dự đoán sẽ tốt nếu nó cho kết quả đúng, nên ta có thể đánh giá chất lượng của giả thuyết bằng cách kiểm tra dự đoán của nó với phân lớp đúng khi ta biết nó. Ta làm điều này trên một tập mẫu được gọi là tập thử nghiệm. Nếu ta huấn luyện trên tất cả mẫu có sẵn thì ta sẽ phải lấy thêm những mẫu khác để thử, thông thường để tiện lợi thì người ta thực hiện phương pháp sau:

1. Thu thập một tập mẫu lớn.
2. Chia nó làm 2 tập phân biệt: **tập huấn luyện** và **tập thử**.
3. Áp dụng thuật toán học trên tập huấn luyện tạo ra một giả thuyết *h*.
4. Đo tỉ lệ phần trăm mẫu trong tập thử được phân lớp đúng bởi h.
5. Lặp lại bước 2 đến 4 cho những tập huấn luyện kích thước khác nhau và với mỗi kích thước tập huấn luyện được chọn ngẫu nhiên

III.6 Nhiễu và quá khớp

Ta thấy trong phần trước nếu hai hay nhiều mẫu có cùng mô tả (tức là các thuộc tính) nhưng khác kết quả phân lớp thì thuật toán sẽ thất bại trong việc tìm ra một cây quyết định nhất quán cho tất cả các mẫu. Giải pháp đề cập trước đây là cho mỗi nút lá hoặc trả về phân lớp cho đa số mẫu hoặc trả về ước lượng xác suất của mỗi phân lớp. Không may, việc này không giải quyết hết vấn đề. Thuật toán học cây quyết định vẫn có thể tìm ra cây quyết định nhất quán cho tất cả các mẫu trong trường hợp thông tin quan trọng bị thiếu. Bởi vì thuật toán có thể dùng những thuộc tính khác nếu có để phân biệt giữa các mẫu. Xem xét việc dự đoán kết quả tung xúc sắc. Giả sử rằng thí nghiệm được tiến hành trong thời gian dài trên nhiều con xúc sắc và các thuộc tính mô tả mỗi mẫu huấn luyện như sau:

1. *Thứ*: ngày thực hiện tung xúc sắc (Thứ 2, Thứ 3, Thứ 4, Thứ 5).

2. *Tháng*: tháng thực hiện tung xúc sắc (Tháng 1 hoặc Tháng 2).
3. *Màu*: màu của xúc sắc (Xanh hoặc Đỏ).

Miễn là không có hai mẫu nào giống nhau, thuật toán học cây quyết định sẽ tìm ra giả thuyết đúng. Càng nhiều thuộc tính thì thuật toán sẽ càng dễ tìm ra giả thuyết đúng. Bất kì giả thuyết nào như vậy sẽ hoàn toàn không chính xác. Điều ta mong muốn là thuật toán học trả về một nút lá với xác suất gần bằng 1/6 cho mỗi lần tung, một khi nó học đủ mẫu.

Khi ta có một tập lớn những giả thuyết có thể thì nên cẩn thận không sử dụng kết quả để tìm ra những "dạng" vô nghĩa trong dữ liệu. Vấn đề này gọi là **quá khớp**. Một hiện tượng rất tổng quát, quá khớp xảy ra ngay cả khi hàm mục tiêu không hề ngẫu nhiên. Nó ảnh hưởng tới mọi loại thuật toán học không chỉ có cây quyết định.

Việc xử lý quá khớp thuần túy bằng toán học vượt quá phạm vi giáo trình này. Ở đây một kỹ thuật đơn giản dùng để giải quyết vấn đề này được gọi là tỉa cây quyết định. Bằng cách tỉa cây, nhiễu có thể được loại bỏ: phân lớp nhầm làm tăng tuyến tính việc dự đoán sai, khi lỗi trong mô tả mẫu có tác động tiệm cận càng xấu đi khi cây thu nhỏ trên tập nhỏ hơn. Cây được xây dựng cùng với việc tỉa thực thi hiệu quả hơn nhiều so với cây xây dựng không qua thao tác tỉa và khi dữ liệu chứa một lượng lớn nhiễu. Cây được tỉa thường nhỏ hơn nhiều và do đó dễ hiểu hơn nhiều.

Kiểm tra chéo là một kĩ thuật khác giúp giảm thiểu quá khớp. Nó có thể được áp dụng cho bất kì thuật toán học nào không chỉ riêng học cây quyết định. Ý tưởng cơ bản là ước lượng mỗi giả thuyết dự đoán dữ liệu chưa học tốt cỡ nào. Việc này được thực hiện bằng cách lấy ra một số phần dữ liệu đã biết và sử dụng nó để kiểm tra khả năng dự đoán của giả thuyết qui nạp từ dữ liệu còn lại. K-fold kiểm tra chéo nghĩa là ta chạy k lần thử, mỗi lần chạy lấy ra 1/k khác nhau của bộ dữ liệu để kiểm tra và tính kết quả trung bình. Giá trị k thường dùng là 5 và 10. Nhỏ nhất là $k = n$ được xem là kiểm tra chéo với một phần tử. Kiểm tra chéo có thể được sử dụng kết hợp với bất kì phương pháp xây dựng cây nào (bao gồm cả tỉa cây) để chọn ra một cây có khả năng dự đoán tốt. Để tránh bị việc quá trình học đã biết trước tập kiểm tra ta phải đo hiệu quả bằng một tập kiểm tra mới.

III.7 Mở rộng tính ứng dụng của cây quyết định

Để mở rộng cây quyết định cho nhiều bài toán khác nhau, một số vấn đề cần được quan tâm sẽ đề cập sơ qua. Các vấn đề quan trọng là:

- Dữ liệu bị thiếu: Trong nhiều lĩnh vực, không phải tất cả giá trị thuộc tính của mỗi mẫu đều được biết trước. Những giá trị này có thể không được ghi nhận hoặc có thể rất tốn kém để có được. Việc này đặt ra hai vấn đề: một là với một cây quyết định đầy đủ, làm thế nào để phân lớp một đối tượng mà nó bị thiếu một trong những thuộc tính cần kiểm tra? Hay là nên thay đổi công thức lượng thông tin có được như thế nào khi một số mẫu có giá trị thuộc tính chưa biết? Những câu hỏi này xem như bài tập.
- Thuộc tính đa giá trị: Khi một thuộc tính có nhiều giá trị, độ đo thông tin không phù hợp để cho biết sự hữu dụng của thuộc tính. Trường hợp xấu nhất, ta có thể dùng một thuộc tính mà có giá trị khác nhau cho mỗi mẫu, chẳng hạn *TênNhàHàng*. Khi đó mỗi tập con sẽ là một đơn thể với một phân lớp duy nhất, nên độ đo thông tin sẽ có giá trị cao nhất cho thuộc tính này. Tuy nhiên, thuộc tính này có thể không liên quan gì hoặc vô dụng. Một giải pháp là sử dụng độ lợi thông tin.
- Thuộc tính đầu vào có giá trị nguyên và liên tục: Thuộc tính có giá trị nguyên hoặc liên tục như chiều cao và cân nặng có tập vô hạn các giá trị có thể có. Thay vì phát sinh vô hạn nhánh, thuật toán học cây quyết định thường tìm điểm phân chia sao cho đạt lượng thông tin cao nhất. Ví dụ, tại một nút trên cây, có thể trường hợp kiểm tra chiều cao>160cm cung cấp nhiều thông tin nhất. Phương pháp qui hoạch động sẽ giúp tìm ra điểm chia tốt, nhưng vẫn còn khá xa so với phần phức tạp nhất của học cây quyết định trong ứng dụng thực tế.
- Thuộc tính đầu ra có giá trị liên tục: Nếu ta định dự đoán một giá trị dạng số, như giá trị một tác phẩm nghệ thuật chẳng hạn, thì ta cần một cây hồi qui. Cây này có mỗi nút lá là một hàm tuyến tính của thuộc tính số thay vì chỉ một giá trị. Ví dụ, nhánh của tác phẩm điêu khắc bằng tay có thể kết thúc với hàm tuyến tính theo diện tích, tuổi và số lượng màu sắc. Thuật toán học phải quyết

định khi nào dừng việc phân chia và bắt đầu áp dụng hồi qui tuyến tính sử dụng những thuộc tính còn lại.

Hệ thống học cây quyết định cho các ứng dụng thực tế phải giải quyết tất cả vấn đề trên. Việc xử lý biến có giá trị liên tục đặc biệt quan trọng, vì các qui trình vật lý và tài chính sử dụng dữ liệu dạng số. Nhiều gói sản phẩm thương mại đã được xây dựng thỏa mãn những tiêu chí này và chúng được dùng để phát triển hàng trăm hệ thống chuyên biệt. Trong nhiều lĩnh vực công nghiệp và thương mại, cây quyết định thường là phương pháp đầu tiên được dùng khi cần một phương pháp phân lớp trên tập dữ liệu. Một tính chất quan trọng của cây quyết định là con người có thể hiểu đầu ra của thuật toán học. Tính chất này không có trong mạng nơ-ron nhân tạo.

IV. KẾT CHƯƠNG

Chương này giới thiệu một số ý tưởng và kỹ thuật cơ bản về học máy trong đó tập trung vào học có giám sát với một số điểm quan trọng như sau:

- Học máy có nhiều dạng khác nhau tùy thuộc vào bản chất của phương pháp và sự hiện diện của phản hồi trong quá trình học.
- Nếu có sự phản hồi trong quá trình học, từ người thầy hay từ môi trường, cho biết các giá trị đúng của các mẫu, bài toán học được gọi là học có giám sát. Nhiệm vụ của học máy lúc này là rút ra từ các mẫu một ánh xạ giữa đầu vào và đầu ra.
- Phương pháp Naïve Bayes là một mô hình học máy khá cơ bản có nền tảng toán học chắc chắn. Tuy nhiên, phương pháp này cũng dựa trên giả định độc lập có điều kiện của các thuộc tính nên hạn chế khả năng áp dụng trên nhiều lớp bài toán.
- Cây quyết định có thể biểu diễn tất cả ánh xạ logic. Độ lợi thông tin cung cấp một phương pháp hiệu quả để tìm một cây đơn giản và nhất quán với dữ liệu. Phương pháp cắt tỉa của Ockham giúp ta rút gọn cây.
- Hiệu quả của một thuật toán học cho biết sự chính xác của dự đoán trên tập thử và phụ thuộc vào kích thước tập huấn luyện.

V. THUẬT NGỮ TIẾNG ANH

Sau đây là những thuật ngữ tiếng Anh liên quan đến những khái niệm đã dùng ở chương này:

classification: phân lớp

conditional independence: độc lập có điều kiện

cross-validation: kiểm tra chéo

decision tree: cây quyết định

information gain: độ lợi thông tin

Laplace correction: phép sửa Laplace

maximum likelihood estimation: ước lượng khả suất tối đại

naïve Bayesian classifier: bộ phân lớp naïve Bayes

noise: nhiễu

Ockham's razor: phương pháp cắt tỉa của Ockham

overfitting: quá khớp

regression: hồi qui

reinforcement learning: học tăng cường

supervised learning: học có giám sát

test set: tập thử

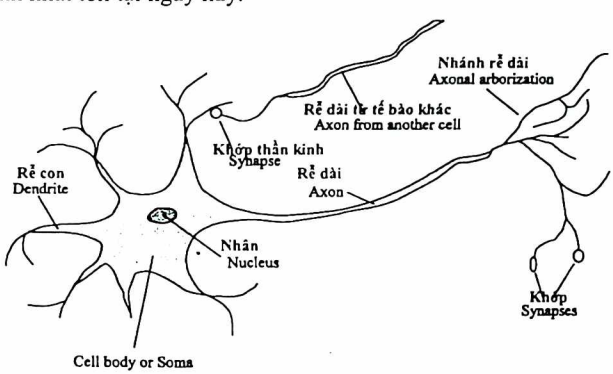
training set: tập huấn luyện

unsupervised learning: học không giám sát

Chương 9
MẠNG NƠ-RON NHÂN TẠO

I. GIỚI THIỆU – MẠNG NƠ-RON LÀ GÌ?

Một **mạng nơ-ron** có thể được định nghĩa là một mô hình suy diễn mô phỏng theo bộ não của con người. Bộ não chứa một tập hợp các tế bào thần kinh, hay là những đơn vị xử lý thông tin cơ bản gọi là nơ-ron, kết nối lẫn nhau một cách dày đặc. Bộ não con người tích hợp gần 10 tỷ nơ-ron và 60 ngàn tỷ kết nối giữa chúng, gọi là các ***khớp thần kinh***. Bằng cách sử dụng nhiều nơ-ron đồng thời, bộ não có thể thực thi chức năng của mình nhanh hơn nhiều lần so với các máy tính nhanh nhất tồn tại ngày nay.



Hình 9.1 - *Các thành phần của một nơ-ron.*

Mặc dù mỗi nơ-ron có một cấu trúc rất đơn giản, một đội quân của các phần tử như thế cũng tạo thành một sức mạnh xử lý khổng lồ. Một nơ-ron bao gồm một **nhân tế bào**, một số **rễ con** và một sợi **rễ dài**. Trong

khi các rễ con phân bố trong một mạng xung quanh một nhân thần kinh, rễ dài kéo dài đến các rễ con và nhân của các nơ-ron khác. Hình 9.1 minh họa một số phần của một nơ-ron.

Các tín hiệu được lan truyền từ một nơ-ron này đến nơ-ron khác bằng các phản ứng điện hóa phức tạp. Các tác nhân hóa học từ các khớp thần kinh gây ra một thay đổi trong thế điện năng của một tế bào. Khi thế điện năng này đạt ngưỡng, một tín hiệu điện được gửi xuống qua các nhân thần kinh. Tín hiệu lan rộng ra và dần dần đi đến các khớp thần kinh, làm tăng hay giảm điện thế của chúng. Tuy nhiên, phát hiện lý thú nhất là một mạng nơ-ron có khả năng thay đổi. Để đáp ứng với một mẫu kích hoạt, các nơ-ron thực hiện các thay đổi dài hạn trong độ lớn của các liên kết giữa chúng. Các nơ-ron cũng có thể hình thành các kết nối mới với các nơ-ron khác. Ngay cả toàn bộ tập hợp các nơ-ron đôi khi cũng có thể được di chuyển từ một nơi này đến nơi khác. Các cơ chế này hình thành cơ sở cho việc học của bộ não.

Bộ não của chúng ta có thể được xem như là một hệ thống xử lý thông tin song song, phi tuyến và có độ phức tạp cao. Thông tin được lưu trữ và xử lý trong một mạng thông tin đồng thời trên toàn bộ mạng thay vì tại một vị trí cụ thể. Nói cách khác, trong một mạng nơ-ron, cả dữ liệu và việc xử lý chúng là toàn cục thay vì cục bộ.

Nhờ vào khả năng thay đổi, các kết nối giữa các nơ-ron đưa đến "câu trả lời đúng" được tăng cường, trong khi những kết nối đưa đến "câu trả lời sai" bị làm yếu đi. Kết quả là các mạng nơ-ron có khả năng học thông qua kinh nghiệm.

Việc học là một đặc tính cơ bản và thiết yếu của các mạng nơ-ron sinh học. Chúng có thể thực hiện điều đó một cách dễ dàng và tự nhiên. Điều này dẫn đến những cố gắng để mô phỏng một mạng nơ-ron sinh học trong máy tính.

Mặc dù một mạng nơ-ron nhân tạo ngày nay bắt chước bộ não con người, cũng giống như việc một chiếc máy bay giấy bắt chước một phản lực siêu thanh, đây cũng có thể xem là một bước tiến. Các mạng nơ-ron nhân tạo có khả năng "học", nghĩa là chúng có thể dùng kinh nghiệm để cải thiện hiệu quả của mình. Khi được cung cấp một số lượng mẫu đủ lớn, các mạng nơ-ron nhân tạo có thể tổng quát hóa đến

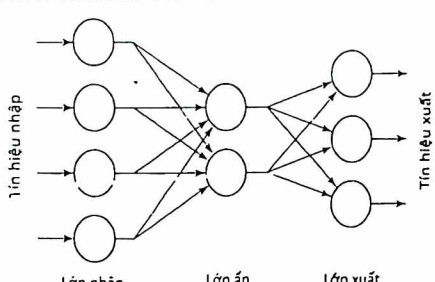
các mẫu chúng chưa hề gặp. Chúng có thể nhận dạng chữ viết tay, nhận diện từ trong lời nói của con người và phát hiện thuốc nổ tại sân bay...

II. MẠNG NƠ-RON NHÂN TẠO

II.1 Kiến trúc mạng nơ-ron nhân tạo

Một **mạng nơ-ron nhân tạo** bao gồm một số các bộ xử lý rất đơn giản và kết nối dày đặc, gọi là các nơ-ron, tương tự như các nơ-ron sinh học trong não. Các nơ-ron được kết nối bởi các liên kết trọng số, truyền các tín hiệu từ một nơ-ron này đến một nơ-ron khác. Mỗi nơ-ron nhận một số tín hiệu nhập (đầu vào) qua các liên kết của nó; tuy nhiên nó không bao giờ tạo ra nhiều hơn một tín hiệu xuất (đầu ra). Tín hiệu đầu ra được chuyển qua các kết nối đi ra ngoài của nơ-ron (tương tự các axon sinh học). Kết nối đi ra ngoài đến lượt nó được chia thành một số nhánh chuyển cùng một tín hiệu (tín hiện không bị chia nhỏ trên các nhánh). Các nhánh đi ra kết thúc tại các kết nối đi vào của các nơ-ron khác trong mạng. Hình 9.2 biểu diễn các kết nối của một mạng nơ-ron nhân tạo tiêu biểu và Bảng 9-1 trình bày sự tương tự giữa các mạng nơ-ron sinh học và nhân tạo.

Các nơ-ron được kết nối bởi các **liên kết** và mỗi liên kết có một **trọng số số học** liên kết với nó. Các trọng số là các trung bình cơ sở của bộ nhớ dài hạn trong các mạng nơ-ron nhân tạo. Chúng biểu diễn độ mạnh kết nối của mỗi đầu vào nơ-ron.



Hình 9.2 - *Kiến trúc của một mạng nơ-ron nhân tạo tiêu biểu.*

Bảng 9-1 *Sự tương tự giữa các mạng nơ-ron sinh học và nhân tạo.*

Mạng nơ-ron sinh học	Mạng nơ-ron nhân tạo
Soma	Nơ-ron
Dendrite	Đầu vào
Axon	Đầu ra
Synapse	Trọng số

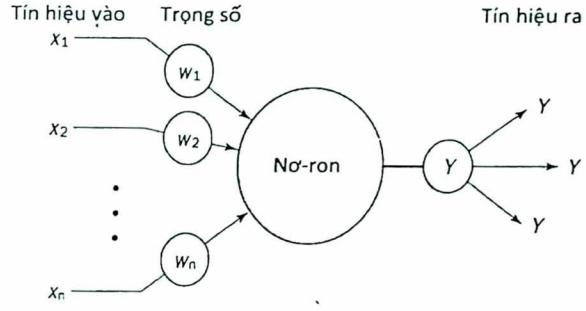
Một mạng nơ-ron "học" bằng cách lặp lại quá trình điều chỉnh các trọng số này.

Như trong Hình 9.2, một mạng nơ-ron nhân tạo được tạo thành theo dạng các lớp phân cấp và các nơ-ron trong mạng được sắp xếp dọc theo các lớp này. Các nơ-ron kết nối với môi trường bên ngoài hình thành nên lớp nhập và xuất. Các trọng số được điều chỉnh để làm cho hành vi nhập/xuất của mạng phù hợp với môi trường. Mỗi nơ-ron là một đơn vị xử lý thông tin cơ sở. Nó có một phương tiện tính toán giá trị kích hoạt dựa trên đầu vào và các trọng số số học.

Để xây dựng một mạng nơ-ron nhân tạo, chúng ta phải quyết định trước tiên sử dụng bao nhiêu nơ-ron và các nơ-ron được kết nối như thế nào để tạo thành mạng. Hay nói cách khác trước tiên chúng ta lựa chọn kiến trúc mạng. Sau đó, ta quyết định sẽ sử dụng thuật toán học nào. Cuối cùng chúng ta huấn luyện mạng nơ-ron, đầu tiên khởi tạo các trọng số của mạng và cập nhật các trọng số từ một tập các mẫu huấn luyện.

II.2 Nơ-ron như là một thành phần tính toán đơn giản

Một nơ-ron nhận một số tín hiệu từ các liên kết đầu vào, tính toán mức độ kích hoạt mới và gởi giá trị này như một tín hiệu đầu ra đến các liên kết đầu ra. Tín hiệu đầu vào có thể là dữ liệu thô hoặc kết xuất của các nơ-ron khác. Tín hiệu đầu ra có thể là lời giải của cuối của vấn đề hoặc một đầu vào cho các nơ-ron khác. Hình 9.3 trình bày một nơ-ron tiêu biểu.



Hình 9.3 - *Sơ đồ của một nơ-ron.*

Nơ-ron tính tổng có trọng số các tín hiệu đầu vào và so sánh kết quả với một giá trị ngưỡng θ. Nếu đầu vào của mạng nhỏ hơn ngưỡng, đầu ra nơ-ron là –1. Nhưng nếu đầu vào mạng lớn hơn hoặc bằng ngưỡng, nơ-ron được kích hoạt và đầu ra của nó mang giá trị +1.

Nói cách khác, nơ-ron dùng hàm chuyển hay **hàm kích hoạt** sau:

$$X = \sum_{i=1}^{n} x_i w_i \tag{9.1}$$

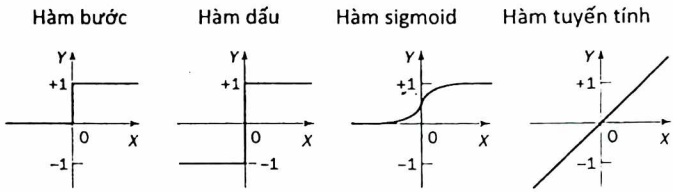
$$Y = \begin{cases} +1 & \text{khi } X \geq 0 \\ -1 & \text{khi } X < 0 \end{cases}$$

trong đó X là đầu vào có trọng số của nơ-ron, x_i là giá trị của đầu vào i, w_i là trọng số của đầu vào i, n là số đầu vào nơ-ron và Y là giá trị đầu ra của nơ-ron.

Loại hàm kích hoạt này được gọi là **hàm dấu**. Đầu ra thật sự của nơ-ron với một hàm kích hoạt dấu có thể được biểu diễn bằng:

$$Y = sign\left[\sum_{i=1}^{n} x_i w_i - \theta\right] \tag{9.2}$$

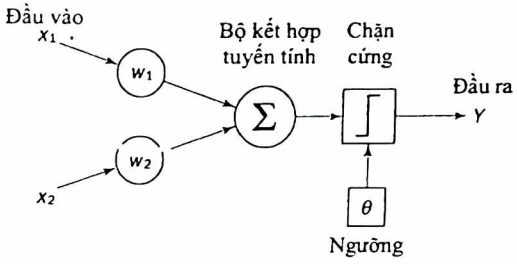
Nhiều hàm kích hoạt đã được kiểm tra nhưng chỉ có một số có ứng dụng thực tế. Bốn lựa chọn phổ biến là hàm bước (*step*), dấu (*sign*), tuyến tính (*linear*) và *sigmoid* được mô tả trong Hình 9.4.



Hình 9.4 - *Hàm kích hoạt của các nơ-ron.*

Các hàm kích hoạt bước và dấu, còn được gọi là các **hàm giới hạn cứng**, thường được sử dụng cho các nơ-ron ra quyết định cho các nhiệm vụ phân lớp và nhận dạng mẫu. Hàm *sigmoid* biến đổi đầu vào, vốn có thể có bất cứ giá trị hữu hạn nào, thành một giá trị tương ứng nằm trong khoảng 0 và 1. Các nơ-ron với hàm này có thể được dùng trong các mạng lan truyền ngược. Hàm kích hoạt tuyến tính cung cấp một đầu ra có giá trị bằng với đầu vào có trọng số của nơ-ron. Các nơ-ron với hàm tuyến tính thường dùng cho việc xấp xỉ tuyến tính.

Năm 1985, *Frank Rosenblatt* giới thiệu một thuật toán cung cấp thủ tục đầu tiên để huấn luyện một mạng nơ-ron nhân tạo đơn giản: một ***perceptron***. *Perceptron* là dạng đơn giản nhất của một nơ-ron nhân tạo. Nó bao gồm một nơ-ron duy nhất với các trọng số **có thể điều chỉnh được** và một **ngưỡng cố định**. Một *perceptron* một lớp hai đầu vào được trình bày trong Hình 9.5.



Hình 9.5 - *Perceptron một lớp hai đầu vào.*

III. PERCEPTRON

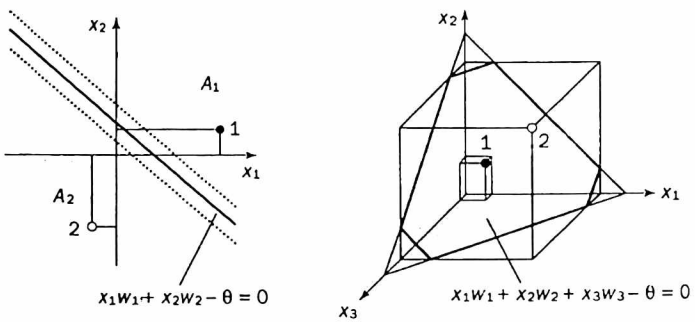
III.1 Các perceptron biểu diễn điều gì?

Hoạt động của một *perceptron* dựa trên mô hình nơ-ron của *McCulloch* và *Pitts*. Mô hình gồm một hàm kết hợp tuyến tính và một ngưỡng cố định. Tổng theo trọng số của các đầu vào sẽ được trừ giá trị ngưỡng và sẽ cho ra kết quả đầu ra bằng +1 nếu đầu vào dương và −1 nếu đầu vào âm. Mục đích của *perceptron* là phân lớp đầu vào, hay nói cách khác ngoại suy các giá trị kích thích: $x_1, x_2, \ldots, x_n$ vào một trong hai lớp: A_1 và A_2. Do đó, trong trường hợp một *perceptron* cơ bản, không gian n-chiều được chia tách bởi một **siêu phẳng** thành hai miền quyết định. Siêu phẳng này được định nghĩa bởi một hàm **khả phân tuyến tính**:

$$\sum_{i=1}^{n} x_i w_i - \theta = 0. \tag{9.3}$$

Đối với trường hợp hai đầu vào x_1 và x_2, biên quyết định có dạng một đường thẳng đậm trong Hình 9.6 (a). Với ba giá trị đầu vào, siêu phẳng vẫn có thể được vẽ ra như trong Hình 9.6 (b). Mặt phẳng phân biệt này được định nghĩa bởi phương trình:

$$x_1 w_1 + x_2 w_2 + x_3 w_3 - \theta = 0.$$



(a) (b)

Hình 9.6 - *Tính phân biệt tuyến tính trong các perceptron: (a) perceptron hai đầu vào; (b) perceptron ba đầu vào.*

Việc xây dựng các phương trình được thực hiện bằng cách áp dụng những thay đổi nhỏ trên các trọng số để làm giảm sự khác biệt giữa các giá trị đầu ra thực tế và đầu ra mong muốn của *perceptron*. Giá trị trọng số ban đầu được gán ngẫu nhiên, thường là trong khoảng [–0.5, 0.5] và sau đó được cập nhật để đạt được đầu ra nhất quán với các mẫu huấn luyện. Đối với một *perceptron*, quá trình cập nhật trọng số đặc biệt đơn giản. Nếu tại vòng lặp thứ p, đầu ra thực tế là $Y(p)$ và đầu ra mong muốn là $Y_d(p)$, thì sai số được cho bởi:

$$e(p) = Y_d(p) - Y(p) \quad \text{trong đó } p = 1, 2, 3, \ldots \tag{9.4}$$

Vòng lặp p ở đây dùng để chỉ mẫu huấn luyện thứ p được biểu diễn bởi *perceptron*.

Nếu sai số $e(p)$ dương, ta cần tăng đầu ra *perceptron* $Y(p)$, nhưng nếu sai số âm, ta cần giảm $Y(p)$. Do mỗi đầu vào *perceptron* đóng góp giá trị $x_i(p) \times w_i(p)$ vào đầu vào tổng $X(p)$, ta thấy rằng nếu giá trị đầu vào $x_i(p)$ là dương, việc tăng trọng số $w_i(p)$ của nó sẽ làm tăng giá trị đầu ra *perceptron* $Y(p)$, trong khi nếu $x_i(p)$ âm, việc tăng $w_i(p)$ sẽ làm giảm $Y(p)$. Do đó **quy tắc học *perceptron*** sau đây có thể được thiết lập:

$$w_i(p+1) = w_i(p) + \alpha \times x_i(p) \times e(p) \tag{9.5}$$

trong đó α là **tốc độ học**, một hằng số dương.

III.3 Thuật toán huấn luyện perceptron

Quy tắc học *perceptron* được đề xuất đầu tiên bởi *Rosenblatt* năm 1960. Dùng quy tắc này ta có thể rút ra thuật toán huấn luyện *perceptron* cho việc phân lớp.

Bước 1: Khởi tạo

Đặt giá trị ngẫu nhiên trong đoạn [–0.5, 0.5] cho các trọng số ban đầu $w_1, w_2, ..., w_n$ và ngưỡng θ.

Bước 2: Kích hoạt

Kích hoạt *perceptron* bằng cách dùng các đầu vào $x_1(p), x_2(p), ..., x_n(p)$ và đầu ra mong muốn $Y_d(p)$. Tính đầu ra thực tế tại vòng lặp $p = 1$

$$Y = step\left[\sum_{i=1}^{n} x_i(p) w_i(p) - \theta\right] \tag{6.6}$$

trong đó n là số đầu vào *perceptron*, và *step* là hàm kích hoạt bước.

Bước 3: Huấn luyện trọng số

Cập nhật các trọng số của *perceptron*

$$w_i(p+1) = w_i(p) + \Delta w_i(p) \tag{6.7}$$

trong đó $\Delta w_i(p)$ là sửa lỗi tại vòng lặp p.

Sửa lỗi trọng số được tính bởi **quy tắc delta**:

$$\Delta w_i(p) = \alpha \times x_i(p) \times e(p) \tag{6.8}$$

Bước 4: Lặp

Tăng vòng lặp p lên một, quay lại Bước 2 và lặp lại quá trình cho đến khi hội tụ.

III.3 Phân tích.

Chúng ta sẽ xem xét một ví dụ đơn giản trong việc huấn luyện *perceptron* để biểu diễn các toán tử logic đơn giản. Bảng chân trị của các toán tử AND, OR và XOR được trình bày trong Bảng 9.2. Bảng biểu diễn tất cả tổ hợp có thể của các giá trị cho hai biến x_1 và x_2 và kết quả của các toán tử. *Perceptron* phải được huấn luyện để phân loại các mẫu đầu vào.

Trước tiên hãy xem xét toán tử AND. Sau khi hoàn tất bước đầu tiên, *perceptron* được kích hoạt bởi một chuỗi bốn mẫu đầu vào đại diện cho một **thế hệ**. Các trọng số *perceptron* được cập nhật sau mỗi lần kích hoạt. Quá trình này được lặp lại cho đến khi các trọng số hội tụ đến một tập giá trị. Kết quả được trình bày trong Bảng 9-3.

Bảng 9-2. *Bảng chân trị các phép toán logic cơ bản.*

Biến đầu vào		AND	OR	XOR
x_1	x_2	$x_1 \cap x_2$	$x_1 \cup x_2$	$x_1 \oplus x_2$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

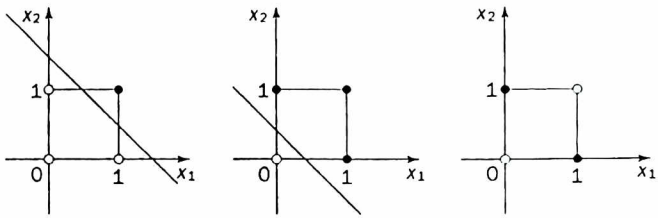
Theo cách tương tự, *perceptron* có thể học được toán tử OR. Tuy nhiên, một *perceptron* một lớp không thể được huấn luyện để phân biệt toán tử XOR. Một minh họa hình học đơn giản có thể giải thích được lý do tại sao. Hình 9.7 biểu diễn các hàm AND, OR và XOR theo các điểm trong không gian 2 chiều dựa trên giá trị của hai đầu vào. Các điểm trong không gian đầu vào mà hàm đầu ra là 1 được thể hiện bằng các điểm đen và các điểm mà hàm đầu ra là 0 được thể hiện bằng các điểm trắng.

Trong các Hình 9.7(a) và (b), ta có thể vẽ một đường thẳng sao cho các điểm đen nằm một phía và các điểm trắng nằm bên phía còn lại nhưng các điểm trong Hình 9.7(c) không thể phân biệt được bởi một đường thẳng. Một *perceptron* có thể biểu diễn một hàm chỉ khi có một đường thẳng có thể phân biệt tất cả điểm đen và điểm trắng. Các hàm

như thế được gọi là **khả phân tuyến tính**. Do đó một *perceptron* có thể học được các toán tử AND và OR nhưng không học được XOR.

Bảng 9-3. *Ví dụ học perceptron: toán tử logic AND.*

Thế hệ	Đầu vào		Đầu ra mong muốn	Trọng số ban đầu		Đầu ra thực tế	Sai số	Trọng số cuối	
	x_1	x_2	Y_d	w_1	w_2	Y	E	w_1	w_2
1	0	0	0	0.3	–0.1	0	0	0.3	–0.1
	0	1	0	0.3	–0.1	0	0	0.3	–0.1
	1	0	0	0.3	–0.1	1	–1	0.2	–0.1
	1	1	1	0.2	–0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	–1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.3	0.0
3	0	0	0	0.3	0.0	0	0	0.2	0.0
	0	1	0	0.3	0.0	0	0	0.2	0.0
	1	0	0	0.3	0.0	1	–1	0.1	0.0
	1	1	1	0.2	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.0	0	0	0.2	0.1
	0	1	0	0.2	0.0	0	0	0.2	0.1
	1	0	0	0.2	0.0	1	–1	0.1	0.1
	1	1	1	0.1	0.0	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1



(a) AND ($x_1 \cap x_2$) (b) OR ($x_1 \cup x_2$) (c) XOR ($x_1 \oplus x_2$)

Hình 9.7 - *Các điểm hai chiều của các toán tử logic cơ bản.*

Việc một *perceptron* chỉ có thể học được các hàm khả phân tuyến tính được suy trực tiếp từ công thức (9.1). Đầu ra *perceptron* Y là 1 chỉ khi tổng đầu vào có trọng số X lớn hơn hoặc bằng giá trị ngưỡng θ. Điều này có nghĩa là toàn bộ không gian đầu vào được chia dọc theo một biên được định nghĩa bởi $X = \theta$. Ví dụ một đường thẳng phân biệt cho đường thẳng AND có thể được định nghĩa bởi phương trình:

$$x_1 w_1 + x_2 w_2 = \theta .$$

Nếu thay thế bằng các giá trị của các trọng số w_1 và w_2 và ngưỡng θ trong Bảng 6.3, ta thu được một trong số các đường thẳng phân biệt có thể là:

$$0.1x_1 + 0.1x_2 = 0.2$$

hay:

$$x_1 + x_2 = 2 .$$

Do đó vùng nằm bên dưới đường thẳng biên, với đầu ra là 0, cho bởi:

$$x_1 + x_2 - 2 < 0 .$$

và vùng nằm trên đường thẳng này, với đầu ra là 1, là:

$$x_1 + x_2 - 2 \geq 0 .$$

Việc một *perceptron* chỉ có thể học được các hàm khả phân tuyến tính là một tin khá xấu vì không có nhiều hàm như thế.

Các *perceptron* một lớp tạo quyết định theo cách tương tự, bất kể chúng sử dụng hàm kích hoạt nào. Điều đó có nghĩa là một *perceptron* một lớp chỉ có thể phân lớp được các mẫu khả phân tuyến tính, bất kể việc chúng ta dùng một hàm kích hoạt chặn cứng hay chặn mềm. Để có thể xử lý các hàm phân biệt tổng quát hơn, ta cần sử dụng các mạng nơ-ron đa lớp. Một trong những mạng nơ-ron đa lớp được sử dụng phổ biến nhất là mạng nơ-ron truyền thẳng đa lớp được huấn luyện bằng thuật toán lan truyền ngược sẽ được giới thiệu trong phần sau.

IV. MẠNG NƠ-RON TRUYỀN THẲNG ĐA LỚP

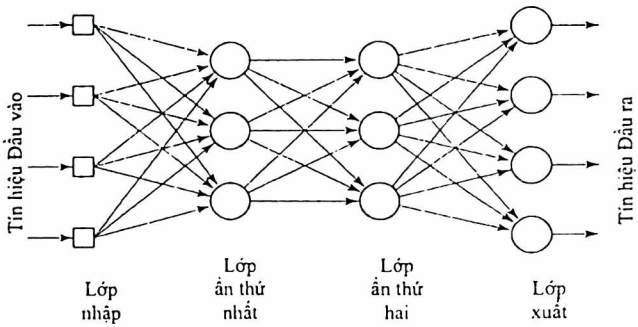
IV.1 Kiến trúc mạng

Một **mạng nơ-ron đa lớp** (hay *perceptron* đa lớp) là một mạng nơ-ron **truyền thẳng** với một hay nhiều lớp ẩn. Thông thường mạng bao gồm một **lớp nhập** các nơ-ron đầu vào, ít nhất một lớp giữa hay **lớp ẩn** các nơ-ron tính toán và một **lớp xuất** các nơ-ron tính toán. Các tín hiệu đầu vào được lan truyền theo một hướng theo từng lớp. Một *perceptron* đa lớp với hai lớp ẩn được trình bày trong Hình 9.8.

Mỗi lớp trong một mạng nơ-ron đa lớp có chức năng cụ thể của mình. Lớp nhập nhận các tín hiệu đầu vào từ thế giới bên ngoài và tái phân phối các tín hiệu này đến tất cả các nơ-ron trong lớp ẩn. Lớp ẩn hiếm khi có các nơ-ron tính toán và do đó lớp này không xử lý các mẫu đầu vào mà thực hiện việc chuyển giao các tín hiệu giữa các lớp xuất và nhập. Lớp xuất nhận các tín hiệu xuất hay nói cách khác là một mẫu kích thích từ lớp ẩn và tạo ra mẫu kết xuất của toàn mạng.

Các nơ-ron trong lớp ẩn xác định các đặc trưng, các trọng số của các nơ-ron biểu diễn các đặc trưng ẩn trong các mẫu đầu vào. Các đặc trưng này sau đó được dùng bởi lớp xuất để xác định mẫu đầu ra. Với một lớp ẩn, ta có thể biểu diễn bất kỳ hàm liên tục theo các tín hiệu đầu vào và với hai lớp ẩn có thể biểu diễn bất kỳ hàm không liên tục.

Các nơ-ron ở lớp giữa "giấu" đầu ra mong muốn của nó nên được gọi là lớp ẩn. Các nơ-ron trong lớp ẩn không thể được quan sát dựa trên hành vi của đầu vào/đầu ra của mạng. Không có cách rõ ràng nào để



Hình 9.8 - *Perceptron đa lớp với hai lớp ẩn.*

biết được đầu ra mong muốn của lớp ẩn là gì. Nói cách khác, đầu ra mong muốn của lớp ẩn được quyết định bởi chính bản thân lớp đấy.

Các mạng nơ-ron thương mại gồm ba hoặc bốn lớp, bao gồm một hoặc hai lớp ẩn. Mỗi lớp có thể chứa từ 10 đến 1000 nơ-ron. Các mạng nơ-ron thực nghiệm có thể có năm hoặc sáu lớp, bao gồm ba hay bốn lớp ẩn và sử dụng hàng triệu nơ-ron nhưng hầu hết các ứng dụng thực tế chỉ dùng ba lớp vì mỗi lớp thêm vào sẽ làm gia tăng đáng kể thời gian tính toán.

IV.2 Thuật toán học lan truyền ngược

Hơn một trăm thuật toán học khác nhau đã được đưa ra nhưng phương pháp phổ biến nhất là **lan truyền ngược**: Học trong một mạng đa lớp diễn ra tương tự như đối với một *perceptron*. Một tập huấn luyện các mẫu đầu vào được đưa vào mạng. Mạng tính toán mẫu đầu ra và nếu có sai số - sai biệt giữa mẫu đầu ra thực tế và mong muốn - các trọng số được điều chỉnh để giảm sai số này.

Trong một *perceptron*, chỉ có một trọng số cho mỗi đầu vào và chỉ một đầu ra. Nhưng trong mạng đa lớp, có nhiều trọng số, mỗi trọng số đều đóng góp cho nhiều hơn một đầu ra.

Trong một mạng nơ-ron lan truyền ngược, thuật toán học có hai pha. Đầu tiên, một mẫu đầu vào huấn luyện được đưa vào lớp nhập của mạng. Mạng lan truyền mẫu đầu vào này theo từng lớp cho đến khi mẫu đầu ra được phát sinh bởi lớp xuất. Nếu mẫu này khác với đầu ra mong muốn, một sai số được tính và sau đó lan truyền ngược lại qua mạng từ lớp xuất đến lớp nhập. Các trọng số được điều chỉnh khi sai số lan truyền.

Thông thường một mạng lan truyền ngược là một mạng đa lớp có ba hay bốn lớp. Các lớp này kết nối đầy đủ với nhau, nghĩa là mọi nơ-ron mỗi lớp đều nối với mọi nơ-ron khác trong lớp lân cận phía trước.

Một nơ-ron xác định giá trị đầu ra của nó theo cách tương tự *perceptron*. Đầu tiên, nó tính đầu vào có trọng số tĩnh như cũ:

$$X = \sum_{i=1}^{n} x_i w_i - \theta$$

trong đó n là số đầu vào và θ là ngưỡng của nơ-ron.

Kế tiếp đầu vào này được truyền qua hàm kích hoạt. Tuy nhiên không như *perceptron*, nơ-ron là một mạng lan truyền ngược dùng hàm kích hoạt sigmoid:

$$Y^{sigmoid} = \frac{1}{1 + e^{-X}} \tag{9.9}$$

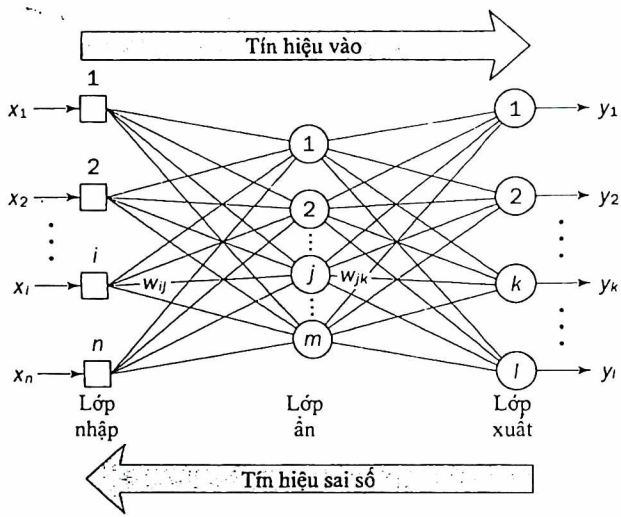
Hàm sigmoid có hai ưu điểm là có thể dễ dàng tính đạo hàm và giá trị đầu ra của nơ-ron sẽ nằm giữa 0 và 1.

Để xây dựng quy tắc học lan truyền ngược, hãy xem xét một mạng ba lớp như trong Hình 9.9. Các chỉ số i, j và k dùng cho các nơ-ron tương ứng trong các lớp nhập, ẩn và xuất. Các tín hiệu đầu vào: $x_1, x_2, ..., x_n$ được lan truyền trong mạng từ trái sang phải và các tín hiệu sai số: e_1, $e_2, ..., e_l$ từ phải sang trái. Ký hiệu w_{ij} biểu diễn cho trọng số kết nối giữa nơ-ron i trong lớp nhập và nơ-ron j trong lớp ẩn và ký hiệu w_{jk} cho trọng số giữa nơ-ron j trong lớp ẩn và nơ-ron k trong lớp xuất.

Để lan truyền sai số, chúng ta bắt đầu tại lớp xuất và truy ngược lại lớp ẩn. Sai số tại đầu ra của nơ-ron k là:

$$e_k(p) = y_{d,k}(p) - y_k(p), \tag{9.10}$$

trong đó $y_{d,k}(p)$ là đầu ra mong muốn của nơ-ron k tại vòng lặp p.



Hình 9.9 - *Mạng nơ-ron lan truyền ngược ba lớp.*

Nơ-ron k, nằm trong lớp xuất, được tính với đầu ra mong muốn của chính nó. Do đó ta có thể dùng thủ tục trực tiếp để cập nhật trọng số w_{jk}. Quy tắc cập nhật trọng số tại lớp xuất tương tự như quy tắc học *perceptron* như trong công thức (9.7):

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p) \tag{9.11}$$

trong đó $\Delta w_{jk}(p)$ là sửa lỗi trọng số.

Khi đã xác định được sửa lỗi trọng số cho *perceptron*, ta dùng tín hiệu đầu vào x_i. Nhưng đối với mạng nhiều lớp, các đầu vào của các nơ-ron lớp xuất khác với đầu vào của nơ-ron lớp nhập.

Khi đó, ta dùng đầu ra của nơ-ron j trong lớp ẩn, y_j, thay cho đầu vào x_i. Sửa lỗi trọng số trong mạng nhiều lớp được tính bởi:

$$\Delta w_{jk}(p) = \alpha \times y_j(p) \times \delta_k(p) \tag{9.12}$$

trong đó $\delta_k(p)$ là gradient sai số tại nơ-ron k trong lớp xuất tại vòng lặp p.

Gradient sai số được xác định là đạo hàm của hàm kích hoạt nhân với sai số tại nơ-ron đầu ra. Với nơ-ron k tại lớp xuất, ta có:

$$\delta_k(p) = \frac{\partial y_k(p)}{\partial X_k(p)} \times e_k(p) \tag{9.13}$$

trong đó $y_k(p)$ là giá trị đầu ra của nơ-ron k tại vòng lặp p, và $X_k(p)$ là giá trị đầu vào có trọng số tĩnh đến nơ-ron k trong cùng vòng lặp.

Đối với một hàm kích hoạt sigmod, biểu thức trên có dạng:

$$\delta_k(p) = \frac{\partial\left\{\dfrac{1}{1+\exp[-X_k(p)]}\right\}}{\partial X_k(p)} \times e_k(p) = \frac{\exp[-X_k(p)]}{\{1+\exp[-X_k(p)]\}^2} \times e_k(p)$$

hay:

$$\delta_k(p) = y_k(p) \times [1 - y_k(p)] \times e_k(p) \tag{9.14}$$

trong đó:

$$y_k(p) = \frac{1}{1+\exp[-X_k(p)]}$$

Để tính sửa lỗi trọng số đối với lớp ẩn, ta có thể áp dụng công thức tương tự như đối với lớp xuất:

$$\Delta w_{ij}(p) = \alpha \times x_i(p) \times \delta_j(p) \tag{9.15}$$

trong đó $\delta_j(p)$ biểu diễn gradient sai số tại nơ-ron j trong lớp ẩn:

$$\delta_j(p) = y_j(p) \times [1 - y_j(p)] \times \sum_{k=1}^{l} \delta_k(p) w_{jk}(p)$$

với l là số nơ-ron trong lớp xuất:

$$y_j(p) = \frac{1}{1 + \exp[-X_j(p)]}$$

$$X_j(p) = \sum_{i=1}^{n} x_i(p) \times w_{ij}(p) - \theta_j$$

và n là số nơ-ron trong lớp nhập.

Tổng kết lại, thuật toán huấn luyện lan truyền ngược đối với mạng nơ-ron đa lớp như sau:

Bước 1: Khởi tạo

Đặt giá trị ngẫu nhiên cho tất cả trọng số và ngưỡng của mạng. Một cách đặt giá trị được đề xuất bởi Haykin năm 1999 là chọn các giá trị ngẫu nhiên phân phối đều trong một khoảng giá trị nhỏ:

$$\left(-\frac{2.4}{F_i}, +\frac{2.4}{F_i} \right)$$

trong đó F_i là tổng số đầu vào của neuron i trong mạng. Việc khởi tạo trọng số được thực hiện cho từng nơ-ron một.

Bước 2: Kích hoạt

Kích hoạt mạng nơ-ron lan truyền ngược bằng cách đưa các giá trị đầu vào và đầu ra mong muốn vào mạng.

(a) Tính đầu ra thực sự của các nơ-ron trong lớp ẩn:

$$y_j(p) = sigmoid\left[\sum_{i=1}^{n} x_i(p) \times w_{ij}(p) - \theta_j \right]$$

trong đó n là số đầu vào của nơ-ron j trong lớp ẩn và *sigmoid* là hàm kích hoạt.

(b) Tính đầu ra thực sự của các nơ-ron trong lớp xuất:

$$y_k(p) = sigmoid\left[\sum_{j=1}^{m} x_{jk}(p) \times w_{jk}(p) - \theta_k\right]$$

trong đó m là số đầu vào của nơ-ron j trong lớp xuất.

Bước 3: Huấn luyện trọng số

Cập nhật trọng số trong mạng lan truyền ngược bằng cách lan truyền ngược sai số ứng với các nơ-ron đầu ra.

(a) Tính gradient sai số cho nơ-ron trong lớp xuất:

$$\delta_k(p) = y_k(p) \times [1 - y_k(p)] \times e_k(p)$$

trong đó:

$$e_k(p) = y_{d,k}(p) - y_k(p)$$

Tính sửa lỗi trọng số:

$$\Delta w_{jk}(p) = \alpha \times y_j(p) \times \delta_k(p)$$

Cập nhật trọng số tại các nơ-ron đầu ra:

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$

(b) Tính gradient sai số cho nơ-ron trong lớp ẩn:

$$\delta_j(p) = y_j(p) \times [1 - y_j(p)] \times \sum_{k=1}^{l} \delta_k(p) w_{jk}(p)$$

Tính sửa lỗi trọng số:

$$\Delta w_{ij}(p) = \alpha \times x_i(p) \times \delta_j(p)$$

Cập nhật trọng số tại các nơ-ron ẩn:

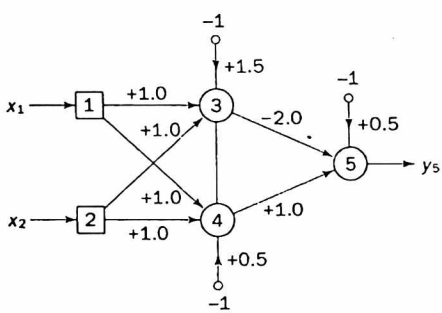
$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

Bước 4: Lặp

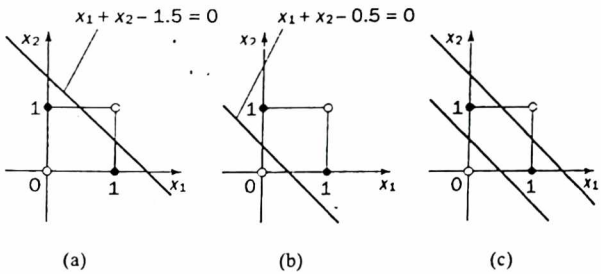
Tăng vòng lặp p lên một, quay lại Bước 2 và lặp lại quá trình cho đến khi điều kiện sai số cho trước thỏa.

Tổng bình phương sai số là một chỉ số hữu ích để đánh giá hiệu quả của mạng. Thuật toán huấn luyện lan truyền ngược cố gắng tối thiểu hoá chỉ số này. Khi giá trị tổng bình phương sai số trong toàn bộ một vòng lặp trên tất cả tập huấn luyện hay một thế hệ **đủ nhỏ**, mạng xem như đã hội tụ.

Hình 9.10 minh họa một mạng nơ-ron ba lớp với năm nơ-ron được huấn luyện để biểu diễn toán tử XOR. Lưu ý là rất khó để vẽ biên quyết định được tạo ra bởi các nơ-ron với một hàm kích hoạt *sigmoid*. Thay vào đó chúng ta có thể biểu diễn mỗi nơ-ron trong lớp ẩn và xuất sử dụng một hàm dấu. Hình 9.11 minh họa biên quyết định cho được tạo ra từ mạng nơ-ron trên. Vị trí của biên quyết định được tạo ra bởi nơ-ron 3 và 4 trong lớp ẩn được biểu diễn trong các Hình 9.11(a) và (b). Nơ-ron 5 trong lớp xuất thực hiện một tổ hợp tuyến tính của các biên quyết định tạo thành bởi các nơ-ron ẩn. Mạng nơ-ron như trong Hình 9.10 thực sự có thể phân biệt được các điểm đen và trắng và do đó có thể giải quyết được bài toán XOR.



Hình 9.10 - *Mạng nơ-ron ba lớp giải toán tử XOR.*



Hình 9.11 - *(a) Biên quyết định được tạo bởi nơ-ron ẩn 3 của mạng trong Hình 9.10; (b) biên quyết định tạo bởi nơ-ron ẩn 4; (c) các biên quyết định tạo bởi toàn bộ mạng ba lớp.*

V. ỨNG DỤNG CỦA MẠNG NƠ-RON

Phần này giới thiệu một vài ví dụ về nhiều ứng dụng đóng vai trò quan trọng của mạng *nơ-ron*. Trong mỗi trường hợp, thiết kế mạng là kết quả của một vài tháng thử nghiệm của các nhà nghiên cứu. Từ những ví dụ này, có thể thấy mạng *nơ-ron* có khả năng ứng dụng rộng rãi, nhưng điều đó không có nghĩa là chúng có thể giải quyết những vấn đề mà không có suy nghĩ của những người thiết kế mạng. *Richard Denker* nhận xét rằng "mạng *nơ-ron* là giải pháp tốt thứ hai trong việc giải quyết bất cứ thứ gì". Điều này cũng phần nào đúng vì mạng *nơ-ron* cung cấp hiệu suất chấp nhận được trong nhiều tác vụ vốn tương đối khó để giải quyết triệt để với những kỹ thuật lập trình khác.

Ứng dụng phát âm

Phát âm một đoạn văn bản tiếng Anh bằng máy tính là một bài toán hấp dẫn trong ngôn ngữ học, cũng như là một công việc có hiệu quả thương mại cao. Đây là ứng dụng tiêu biểu nhằm chuyển từ văn bản sang **tiếng nói** – yếu tố âm thanh cơ bản – và sau đó truyền tiếng nói đến một máy phát. Vấn đề chúng ta quan tâm ở đây là việc học ánh xạ từ văn bản đến tiếng nói. Đây là một công việc tốt của mạng nơ-ron bởi vì hầu như các quy tắc chỉ gần đúng. Chẳng hạn, mặc dù mẫu tự

"k" thường tương ứng với phát âm k, thì mẫu tự "c" vẫn được phát âm là [k] trong từ *car* và [s] trong từ *cent*.

Chương trình NETtalk của *Sejnowski* và *Rosenberg* năm 1987 là một mạng nơ-ron học phát âm từ những văn bản. Đầu vào là một chuỗi các kí tự hiện diện trong một cửa sổ duyệt qua văn bản. Bất cứ lúc nào, đầu vào bao gồm những kí tự được phát âm cùng với 3 kí tự trước và sau. Mỗi kí tự thật ra là 29 đơn vị đầu vào – một cho mỗi một trong 26 chữ cái và cho mỗi khoảng trống, mỗi dấu chấm câu và các phép chấm câu khác. Có khoảng 80 đơn vị ẩn. Lớp đầu ra được tổ chức theo từng lớp bao gồm đặc trưng của âm thanh được tạo ra: cao hay thấp, âm hay không âm,...

Trên dữ liệu kiểm tra, độ chính xác của NETtalk khoảng 78%, kém hơn nhiều so với những chương trình được đưa vào thương mại. Dĩ nhiên, các hệ thống thương mại yêu cầu đã phát triển qua nhiều năm, trong khi đó, NETtalk chỉ yêu cầu vài giờ huấn luyện và vài tháng kiểm nghiệm với các thiết kế mạng khác nhau.

Nhận dạng chữ viết tay

Đây là một trong những ứng dụng rộng rãi nhất của mạng nơ-ron đến ngày nay. Năm 1989, *Le Cun* và các đồng nghiệp đã thực hiện một mạng được thiết kế để đọc các mã bưu điện trên thư từ được gửi bằng tay. Hệ thống sử dụng một bộ tiền xử lý định vị và phân đoạn các con số riêng biệt trong mã bưu điện; mạng phải xác định các con số của chúng. Mạng được huấn luyện trên 7300 mẫu và được kiểm tra trên 2000 mẫu. Một tính chất thú vị của một mạng với sự mã hóa phân tán đầu ra là nó có thể biểu diễn sự nhập nhằng trong các dự đoán bằng việc xuất ra 2 hay nhiều đơn vị đầu ra lớn hơn một ngưỡng. Sau khi loại bỏ khoảng 12% tập kiểm tra nằm gần ngưỡng nhập nhằng này, hiệu suất cho những trường hợp còn lại đạt 99% và được cho rằng thích hợp với hệ thống sắp xếp mail tự động. Mạng cuối cùng được bổ sung trong VLSI truyền thống, có thể giúp các chữ cái được sắp xếp với một tốc độ cao.

Ứng dụng trong lái xe

ALVINN (Autonmous Land Vehicle In a Neuron Network) của *Pomerleau* năm 1993 là một mạng nơ-ron biểu diễn khá tốt ở những

trường hợp mà các phương pháp khác đã thất bại. Mạng học để lái một chiếc xe dọc theo một làn đường trên đường cao tốc bằng cách quan sát thao tác của một người lái xe.

ALVINN được sử dụng để điều khiển xe NavLab tại đại học *Carnegie Mellon*. *NavLab 1* là một chiếc *Chevy van* và *NavLab 2* là một chiếc *U.S Army HMMWV*. Cả hai loại xe đều được điều khiển bằng các máy tính. Các bộ cảm biến bao gồm truyền hình màu lập thể, bộ tìm kiếm quét các dải *laser*, *radar* và cảm biến gia tốc. Các nhà nghiên cứu đã đi kèm chiếc xe và quan sát quá trình chiếc xe tự động lái như thế nào.

Tín hiệu từ chiếc camera quay chiếc xe được tiền xử lý tạo ra một mảng các giá trị pixel được liên kết với lưới kích thước 30×32 các đơn vị đầu vào trong mạng nơ-ron. Giá trị đầu ra là một lớp gồm 30 đơn vị, mỗi đơn vị tương ứng với một hướng lái xe. Đơn vị đầu ra với giá trị kích hoạt cao nhất là hướng là chiếc xe sẽ lái. Mạng cũng có một lớp gồm 5 đơn vị ẩn mà được liên kết đầy đủ với các lớp đầu vào và đầu ra.

Công việc của ALVINN là tính toán một hàm để tính từ một ảnh video đơn lẻ của con đường phía trước nó ra một hướng lái xe. Để học hàm này, chúng ta cần một số dữ liệu huấn luyện – một số cặp hình ảnh/hướng với hướng chính xác. Dữ liệu này được thu bằng việc một người lái xe và ghi nhận lại những cặp hình ảnh/hướng. Sau khi thu khoảng năm phút các dữ liệu huấn luyện (và áp dụng thuật toán lan truyền ngược trong khoảng 10 phút), ALVINN đã sẵn sàng để lái.

Các kết quả của việc huấn luyện là rất ấn tượng. ALVINN đã được lái với tốc độ lên đến 120km/h cho một quãng đường lên đến 145km trên đường cao tốc gần Pittsburgh. Nó cũng được lái với tốc độ trung bình trên các làn đường đơn bùn nhão, đường lát gạch dành cho xe đạp, và hai con đường nhỏ ở ngoại ô.

VI. KẾT CHƯƠNG

Chương này giới thiệu mạng nơ-ron nhân tạo và thảo luận những ý tưởng cơ bản trong việc mô phỏng hoạt động của con người trong học máy. Một số điểm quan trọng của chương bao gồm:

- Các mạng nơ-ron nhân tạo bao gồm một số các đơn vị xử lý đơn giản và kết nối chặt chẽ với nhau gọi là các nơ-ron, tương tự các nơ-ron sinh học của não bộ. Các nơ-ron được kết nối bởi các liên kết có trọng số dùng để truyền các tín hiệu từ một nơ-ron này đến nơ-ron khác.
- Mỗi liên kết có một trọng số số học gắn với nó. Các trọng số là các phương tiện cơ sở của bộ nhớ dài hạn trong các mạng nơ-ron nhân tạo. Chúng biểu thị sức mạnh, hay độ quan trọng, của mỗi nơ-ron đầu vào. Một mạng nơ-ron học bằng cách lặp lại việc điều chỉnh các trọng số này.
- *Perceptron* là mạng nơ-ron đơn giản nhất chỉ bao gồm một nơ-ron duy nhất với các trọng số có thể điều chỉnh và một ngưỡng cố định. *Perceptron* học bằng cách thực hiện những điều chỉnh nhỏ trong các trọng số để làm giảm sai biệt giữa kết xuất mong muốn và thực tế. Các trọng số ban đầu được khởi tạo ngẫu nhiên và được cập nhật để đạt được kết xuất nhất quán với các mẫu huấn luyện.
- Một *perceptron* chỉ có thể học được các hàm khả phân tuyến tính.
- Một *perceptron* đa lớp là một mạng nơ-ron truyền thẳng với một lớp nhập các nơ-ron nguồn, ít nhất một lớp giữa hay lớp ẩn với các nơ-ron tính toán, một lớp xuất với các nơ-ron tính toán.
- Lớp nhập nhận các tín hiệu nhập từ thế giới bên ngoài và phân phối những tín hiệu này đến tất cả các nơ-ron trong lớp ẩn. Lớp ẩn phát hiện các đặc trưng. Các trọng số của nơ-ron trong lớp ẩn biểu diễn các đặc trưng trong mẫu nhập. Lớp xuất xây dựng mẫu xuất của toàn mạng.
- Học trong mạng đa lớp tương tự như trong *perceptron*. Thuật toán học có hai pha. Đầu tiên một mẫu nhập huấn luyện được đưa vào lớp nhập của mạng. Mạng lan truyền mẫu nhập qua các lớp cho đến khi mẫu xuất được phát sinh bởi lớp xuất. Nếu có khác biệt so với kết quả mong muốn, một sai số được tính

toán và lan truyền ngược qua mạng từ lớp xuất về lớp nhập. Các trọng số được điều chỉnh khi sai số được lan truyền.

VII.THUẬT NGỮ TIẾNG ANH

Sau đây là những thuật ngữ tiếng Anh liên quan đến những khái niệm đã dùng ở chương này:

activation function: hàm kích hoạt

adjustable: có thể điều chỉnh được

artificial neural network: mạng nơ-ron nhân tạo

axon: rễ dài

back-propagation: lan truyền ngược

delta rule: quy tắc delta

dendrite: rễ con

epoch: thế hệ

error gradient: gradient sai số

feedforward: truyền thẳng

hard limit function: hàm giới hạn cứng

hidden layer: lớp ẩn

hyperplane: siêu phẳng

input layer: lớp nhập

learning rate: tốc độ học

linearly separable: khả phân tuyến tính

link: liên kết

multilayer perceptron: mạng perceptron đa lớp

neural network: mạng nơ-ron

numerial weight: trọng số số học

output layer: lớp xuất

perceptron: một mạng nơ-ron nhân tạo đơn giản

sign function: hàm dấu

soma: nhân tế bào

synapse: khớp thần kinh

sum of the squared errors: tổng bình phương sai số

TÀI LIỆU THAM KHẢO

1. Stuart Jonathan Russell, Peter Norvig. *Artificial Intelligence: A Modern Approach*. (Third edition). 2010. Publisher Prentice Hall.
2. David L. Poole, Alan K. Mackworth. *Artificial Intelligence: Foundations of computational agents*. Cambridge University press. 2010.
3. Max Lungarella, Lungarella, Josh Bongard, Max Lungarella. *50 Years of Artificial Intelligence: Essays Dedicated to the 50th Anniversary of Artificial Intelligence*. Published January 28th 2008 by Springer.
4. Artificial Intelligence as taught in: Fall 2011. http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/
5. Techniques in Artificial Intelligence (SMA 5504) as taught in: Fall 2002. http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-825-techniques-in-artificial-intelligence-sma-5504-fall-2002/

Tác giả:

Lê Hoài Bắc
PGS.TS – Trưởng Bộ môn KHMT.
Hướng nghiên cứu: Trí tuệ nhân tạo, Tính toán mềm và Khám phá tri thức & Khai thác dữ liệu.

Tô Hoài Việt
Thạc sỹ - Phó trưởng Bộ môn KHMT. Hiện là NCS tại Viện Khoa học và Công nghệ Tiên tiến Nhật bản (JAIST).
Hướng nghiên cứu: Trí tuệ nhân tạo, Học máy, Web ngữ nghĩa.

CHỈ MỤC THUẬT NGỮ

bài toán 8 puzzle, 53, 56
bài toán 8 quân hậu, 16, 42, 60
bài toán 8-puzzle, 14, 15, 16
bài toán bố trí mạch điện tử, 19
bài toán dây chuyền lắp ráp tự động, 19
bài toán giản lược, 56, 57, 71
bài toán người bán hàng, 19
bài toán người đi du lịch, 18
bài toán robot tìm đường, 19
bài toán thỏa mãn ràng buộc, 72
bài toán thực tế, 15, 18
bài toán tìm đường đi, 18
bài toán tìm kiếm, 11, 12, 14, 15, 19, 20, 27, 46, 72, 73, 76
bài toán tô màu, 72
bài toán tối ưu hóa, 59
bài toán trò chơi, 15
bảng chân trị, 86, 87, 88, 89, 99, 106, 107, 186
biến, 72
biến hóa, 159
biểu thức, 103
biểu thức cơ sở, 105, 123, 124, 125, 126
bộ phân lớp Naïve Bayes, 180
câu nguyên tố, 104
câu nguyên tố cơ sở, 104
cây quyết định, 184, 185, 186, 190
cây tìm kiếm, 20
chi phí, 56
chi phí đường đi, 12, 14, 15, 17, 18, 23, 28, 29, 74
chứng minh phản chứng, 91
con trỏ quay lui, 26
công thức trạng thái đầy đủ, 74
cú pháp, 83
cực đại cục bộ, 61, 63
cực đại toàn cục, 59
cực tiểu toàn cục, 59, 60
dạng hội chuẩn, 90, 140
Datalog, 132
độ đo không tin cậy, 164
độ đo tin cậy, 164
độ lợi thông tin, 192
độ phức tạp về không gian, 31
độ phức tạp về thời gian, 30, 31, 33, 47

đồ thị, 12, 13, 20, 22, 23, 44, 73, 95, 96, 139, 148

đồ thị không gian trạng thái, 31, 37, 43, *Xem* đồ thị trạng thái

đồ thị tìm kiếm, 20

đồ thị trạng thái, 12

độc lập có điều kiện, 163, 180, 181

đồng nhất, 128

đột biến, 67

giá trị ràng buộc ít nhất, 79

gradient sai số, 214

hàm đánh giá, 48

hàm đồng nhất tổng quát nhất, 129

hàm heuristic, 49

hàm heuristic chấp nhận được, 51

hàm kích hoạt, 202

hàm mục tiêu, 59, 72

hàm thích nghi, 66

hàm trạng thái con, 15, 20, 47

hàng đợi, 24

hàng đợi First-In-First-Out (FIFO), 26

hàng đợi Last-In-First-Out (LIFO), 26

hàng đợi ưu tiên, 26, 28

hằng Skolem, 125

hệ số chắc chắn, 153, 154, 163, 164, 165, 167, 168, 169, 170, 171

hệ số phân nhánh, 31, 41, 54, 55, 56, 76, 79

heuristic chấp nhận được, 51, 53, 54, 56, 57

heuristic theo bậc, 79

học có giám sát, 176, 178

học không giám sát, 177

học tăng cường, 177

hồi qui, 184

hợp giải, 89, 90, 91, 92, 93, 98, 140, 142, 143, 144, 145, 146, 147, 148

hợp lệ, 72

khả phân tuyến tính, 204, 208

khoảng cách Manhattan, 54

không gian trạng thái, 11, 12, 17, 18, 20, 22, 23, 36, 38, 41, 42, 43, 46, 48, 59, 61, 62, 63, 67

kiểm tra xuất hiện, 130

lai ghép, 67

lan truyền ngược, 211

lập trình logic, 130, 134, 136, 137, 150

leo đồi làm lại ngẫu nhiên, 63

liên kết logic, 85
logic, 83
logic bậc nhất, 84, 100
logic mệnh đề, 84
logic mờ, 153
lời giải, 14, 72
luật Bayes, 161, 179
lược đồ, 68
lượng từ, 105
lượng từ phổ quát, 105, 106, 123, 124, 126, 128, 131, 134, 141, 142
lượng từ tồn tại, 107, 108, 123, 125, 126, 131, 141, 142
lý thuyết chứng minh, 84
mạng nơ-ron, 196, 198, 199, 200, 201, 203, 210, 215, 218, 219, 220, 223
mạng nơ-ron nhân tạo, 200
mạng nơ-ron truyền thẳng đa lớp, 210
mệnh đề hóa, 126
mệnh đề Horn, 93
mệnh đề xác định bậc nhất, 131
miền xác định, 72
mô hình, 100, 101, 102, 204
một mạng nơ-ron nhân tạo, 201
mức độ tin cậy, 153
ngăn xếp, 26, 35
ngữ nghĩa, 83, 85, 102, 104
nhánh non, 24
nhiễu, 193
nút tìm kiếm, 21
phân lớp, 184
phân phối xác suất, 157
phân phối xác suất kết hợp, 157
phép gán, 72
phép sửa Laplace, 183
phức tạp về mặt không gian, 32
phương pháp cắt tỉa của Ockham, 187
phương pháp tìm kiếm, 11
quá khớp, 193, 194
quan hệ hai ngôi, 101
quan hệ một ngôi, 101
ràng buộc, 72
ràng buộc bậc hai, 75
ràng buộc bậc nhất, 74
ràng buộc sở thích, 75
ràng buộc tuyệt đối, 75
Skolem hóa, 125, 142
sự không chắc chắn, 151

sự kiện nguyên tố, 155, 156, 157, 159, 160, 172

suy diễn hướng đích, 96

suy diễn hướng dữ liệu, 95

suy diễn lùi, 93, 96, 97, 98, 134, 135, 136, 138, 144

suy diễn tiến, 93, 94, 95, 96, 130, 131, 132, 133, 134, 136, 139, 140, 147

tập huấn luyện, 193

tập thử, 193

thể hiện, 69

thể hiện phổ quát, 124, 125

thể hiện tồn tại, 125

thuật giải di truyền, 48, 66, 68, 69, 70

thuộc tính, 103

tìm kiếm A*, 51

tìm kiếm chi phí đồng nhất, 27, 28, 29, 31, 33, 44, 47

tìm kiếm có thông tin, 48

tìm kiếm cục bộ, 48, 58, 59, 60, 74

tìm kiếm cục bộ tham lam. *Xem* tìm kiếm leo đồi

tìm kiếm không có thông tin, 11

tìm kiếm kiểu tham lam, 50

tìm kiếm lặp sâu dần, 39, 41, 44, 46, 55, 56

tìm kiếm leo đồi, 60

tìm kiếm leo đồi có lựa chọn, 63

tìm kiếm leo đồi ngẫu nhiên, 63

tìm kiếm mô phỏng luyện kim, 48, 64, 71

tìm kiếm mù. *Xem* tìm kiếm không có thông tin

tìm kiếm nhóm cục bộ, 64

tìm kiếm quay lui, 36, 77

tìm kiếm theo chiều rộng, 26, 27, 28, 31, 32, 33, 36, 39, 41, 44, 46

tìm kiếm theo chiều sâu, 34, 35, 36, 37, 38, 39, 43, 44, 46, 51, 74, 76, 136, 139

tìm kiếm tối ưu, 48

tìm kiếm tối ưu kiểu tham lam, 49

tìm kiếm với độ sâu giới hạn, 38

tìm kiếm với độ sâu tăng dần. *Xem* tìm kiếm lặp sâu dần

tính đầy đủ, 30

tính tối ưu, 30

tính tổng ngoài, 159

tổng bình phương sai số, 217

trạng thái, 12, 14, 15, 17, 18, 23, 67

trạng thái ban đầu, 12, 15, 17, 18, 20, 21, 23, 26, 31, 37, 46, 48, 54, 63, 73

trạng thái con, 12, 14, 17, 18, 20, 30, 31, 42, 46, 60, 61, 62, 63, 64, 66, 73, 77

trạng thái đích, 12, 15, 17, 18, 21, 23, 26, 46, 48, 63, 74

trạng thái hiện tại, 58

ước lượng khả suất tối đại, 182

ước lượng khả suất tối đại, 179

vị từ, 103

xác suất, 156

xác suất có điều kiện. *Xem* xác suất hậu nghiệm

xác suất hậu nghiệm, 158

xác suất không điều kiện. *Xem* xác suất tiên nghiệm

xác suất tiên nghiệm, 157

LÊ HOÀI BẮC – TÔ HOÀI VIỆT

CƠ SỞ TRÍ TUỆ NHÂN TẠO

Chịu trách nhiệm xuất bản: PHẠM NGỌC KHÔI
Biên tập: TRƯƠNG THANH SƠN
PHẠM THỊ MAI
Sửa bài: PHẠM THỊ MAI
Vẽ bìa: VŨ QUỐC HOÀNG

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
70 Trần Hưng Đạo – Quận Hoàn Kiếm – Hà Nội
CHI NHÁNH NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
28 Đồng Khởi, 12 Hồ Huấn Nghiệp – Quận 1 – Tp.HCM
Điện thoại: (08) 3822 5062 – 3829 6628 – 3829 0228

In 1000 cuốn, khổ 16 x 24 cm, tại Xưởng in Trường ĐH Khoa học Tự nhiên ĐHQG Tp Hồ Chí Minh.
Số đăng KH xuất bản: 936 - 2013/CXB/19 - 62/KHKT. Quyết định xuất bản số 285/QĐXB-NXBKHKT, cấp ngày 26 tháng 12 năm 2013.
In xong và nộp lưu chiểu tháng 02 năm 2014.

TÓM TẮT GIÁO TRÌNH

Với sự phát triển vượt bậc của công nghệ, máy tính cùng các thiết bị điện tử ngày càng đóng vai trò quan trọng trong mọi hoạt động của con người. Không chỉ hỗ trợ trong công việc, các máy tính ngày nay còn là một phương tiện hữu ích phục vụ cho các sinh hoạt hàng ngày của con người như hỗ trợ trong đi lại, cập nhật tin tức, giải trí,... Tuy nhiên, để máy tính thật sự trở thành một người bạn thân thiết của con người, một số trở ngại cơ bản cần phải được vượt qua. Các máy tính ngày nay có sức mạnh vượt trội so với trước, khả năng xử lý và lưu trữ khổng lồ, nhưng nó vẫn chưa giải quyết được nhiều bài toán trên thực tế như: làm sao để có thể tự suy nghĩ và hành động trong những môi trường thế giới thực phức tạp, làm sao để học và rút trích các thông tin, tri thức từ môi trường xung quanh phục vụ cho hoạt động của mình.... Giải quyết các vấn đề trên không chỉ đòi hỏi sức mạnh công nghệ mà còn cần đến những hướng tiếp cận, những thuật toán thông minh hơn.

Giáo trình này cung cấp các ý tưởng, kiến thức cơ bản về hướng nghiên cứu trí tuệ nhân tạo, ngành học nhằm mục tiêu thiết kế ra các phương pháp tính toán thông minh. Nội dung của giáo trình sẽ tập trung giải quyết những vấn đề cơ bản của của một hệ thống thông minh bao gồm: Các phương pháp tìm kiếm hiệu quả, Biểu diễn tri thức và Lập luận, Học máy. Các nội dung trình bày trong giáo trình được tham khảo từ những giáo trình, tài liệu về trí tuệ nhân tạo phổ biến trên thế giới. Do đó các kiến thức trong giáo trình phần nào cập nhật với những bước phát triển gần đây trong lĩnh vực nghiên cứu. Với giáo trình này, chúng tôi hy vọng cung cấp cho người đọc một cái nhìn toàn cảnh và cơ bản về những nghiên cứu liên quan đến trí tuệ nhân tạo. Qua đó, có thể động viên, khuyến khích những ai có niềm đam mê nghiên cứu, sáng tạo tiếp tục tìm hiểu và phát triển các hướng chuyên sâu của ngành học.

213266-B00
ISBN 9786046700790

9 786046 700790

35.000 đ